

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Новое
в жизни,
науке,
технике

Подписная
научно-
популярная
серия

Издается
ежемесячно
с 1988 г.

Уроки программирования

1989

10

Новое
в жизни,
науке,
технике

Подписная
научно-
популярная
серия

10/1989

Издается
ежемесячно
с 1988 г.

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

УРОКИ ПРОГРАММИРОВАНИЯ

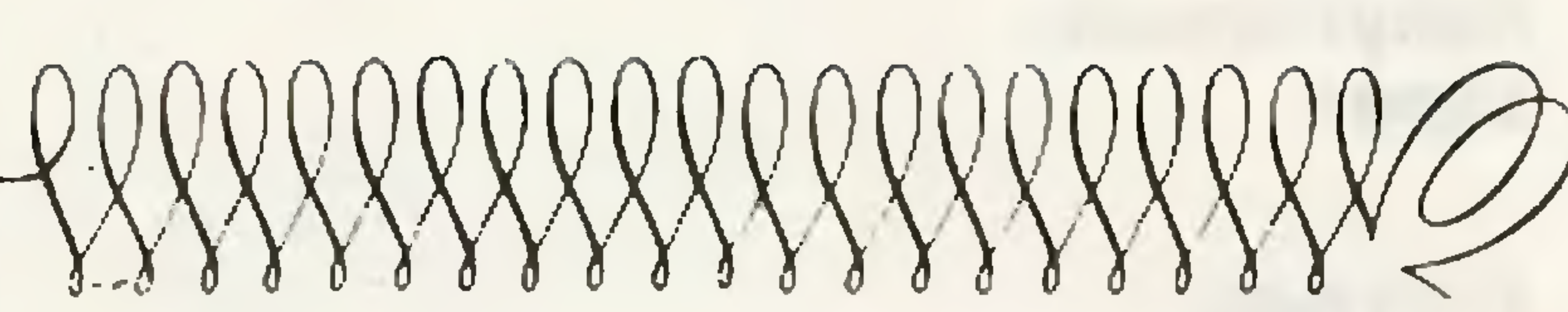
В НОМЕРЕ:



Издательство
«Знание»
Москва
1989

ББК 32.973-01
У 71





ДАГЕНЕ Валентина Антановна — младший научный сотрудник Института математики и кибернетики АН Литовской ССР.

МЕШКОВ Николай Алексеевич — кандидат технических наук, старший научный сотрудник кафедры кибернетики Московского института электронного машиностроения. Руководит студенческим научно-производственным отрядом математиков-программистов «Прометей».

УШАНОВ Сергей Николаевич — известный публицист. Основная направленность публикаций — пути развития научно-технического прогресса.

ОСИПОВ Лев Александрович — кандидат технических наук, доцент, специалист по информатике и вычислительной технике. Имеет 90 научных трудов, из них 60 печатных и 3 изобретения.

КОВАЛЬСКИЙ Илья Семенович — член Союза журналистов СССР, редактор-консультант Центрального телевидения, автор ряда телевизионных программ, связанных с пропагандой компьютерных знаний. Награжден бронзовой, серебряной и золотой медалями ВДНХ.

СЕРГЕЕВ Сергей Владимирович — инженер по электронике, автор нескольких научных статей в журналах. Специалиризируется в области АСУ.

РЕДАКТОР: Б. М. ВАСИЛЬЕВ

ISBN 5-07-000923-0



Здесь представляется краткий курс начал программирования в виде уроков. Цель уроков — ознакомить читателя с основными понятиями программирования, со структурами управления, научить его составлять программы для небольших задач на языке ПАСКАЛЬ. Этот язык универсален и довольно прост, его легко усвоить. Программирование — это творческая работа. Нельзя научиться программировать без практической работы. Поэтому в конце каждого урока приводятся задачи для повторения. Попробуйте их решить, если не удается — еще раз прочитайте урок. Решения сверяйте с ответами (ответы приведены только на задачи, помеченные звездочкой).

В. ДАГЕНЕ

ТРИНАДЦАТЬ УРОКОВ ПРОГРАММИРОВАНИЯ

Рассматриваемые здесь уроки программирования трехкратно публиковались в газете «Комъяунимо тиеса» и использовались как учебный материал для слушателей Литовской заочной школы молодых программистов. На их основе в 1989 г. была выпущена книжка «Учимся программировать»*. Автор благодарит Дайве Дульскене за помощь при подготовке перевода этой книжки.

Первый урок
ЦЕЛЫЕ ЧИСЛА

Компьютер выполняет действия с данными. Они записываются в его память, которую можно изобразить как лист бумаги, поделенный на клеточки. В память заносятся данные, например по одному числу в каждую клеточку. Когда в ту же клеточку заносится новое число, то ранее записанное стирается.

Компьютер читает данные, имеющиеся в памяти (в клеточках), производит с ними нужные действия, и результаты вновь записываются в память. Какие действия и с какими данными выполнять, где записывать результат — указывается в программе.

Данные могут быть разные. Это числа (целые и дробные), буквы, текст — в зависимости от решаемой задачи. Мы будем использовать лишь один тип данных — целые числа. Поэтому будем говорить просто — «число», подразумевая, что это целое число.

Целые числа понимаются так, как в математике. Язык ПАСКАЛЬ позволяет выполнять с целыми числами четыре основные арифметические операции (действия): сложение, вычитание, умножение и деление. Сложение и вычитание обозначаются обычно: + и −. Знак умножения — звездочкой (*). Обратим внимание, что знак умножения никоим образом нельзя пропускать, как это принято делать в математике.

При действии с точностью до целых чисел получаются два результата — частное и остаток. Например, при делении 14 на 5 получается частное 2 и остаток 4. При решении задач иногда требуется только частное, а иногда — только остаток. Язык ПАСКАЛЬ имеет два вида деления: **div** и **mod.** Если надо получить частное, операция обозначается **div,** а если остаток — **mod.**

Например:

13 **div** 5 = 2,	3 **div** 8 = 0,
13 **mod** 5 = 3,	3 **mod** 8 = 3,
5 **div** 5 = 1,	0 **div** 7 = 0,
5 **mod** 5 = 0,	0 **mod** 7 = 0.

Как принято в математике, делить на нуль нельзя. При отрицательных числах операция **div** выполняется просто: числа делятся без учета знаков (положительные). Знак полученного результата — частного — определяется, как и в математике, при делении или умножении, например:

−8 **div** (−3) = 2,
8 **div** (−3) = −2.

Остаток отрицательных чисел представить труднее, поэтому операцию **mod** будем применять лишь к положительным числам.

Выражения образуются, как и в математике. Используются лишь обычные скобки. Порядок операций ничем не отличается от принятого в математике: прежде всего выполняются действия в скобках, затем — умножение и деление, а позднее — сложение и вычитание. Операции одного ранга (например, умножение и оба вида деления)

* Dagienė V. Mokomės programuoti. — Kaunas: Šviesa, 1989; «Sviesos» leidykla, 1989.

выполняются по порядку слева направо. Например, выражение

10 **div** 6 **mod** 3 + 4 * 3 **mod** 5

читается так, как при наличии следующих скобок:

((10 **div** 6) **mod** 3) + ((4 * 3) **mod** 5).

Его значение равно 3.

Приведем несколько примеров выражений:

a) 8 **div** 6 **mod** 4 * 2;
b) 8 **div** (6 **mod** 4) * 2;
c) 8 **div** 6 **mod** (4 * 2);
d) 8 **mod** (6 **mod** 4 * 2);
e) 18 **mod** 7 * 2 — 3 **mod** 7.

Определите значения этих выражений. Результаты должны быть следующие: a) 2; b) 8; c) 1; d) 0; e) 5.

Задачи для повторения

1. Определите значения следующих выражений:

a)* 6 **mod** 7 + 5 **div** 4 * 2;
b)* 6 **mod** (7 + 5) **div** 4 * 2;
c)* 6 **mod** ((7 + 5) **div** 4) * 2;

2. С какими значениями *x* следующие равенства будут правильными?

a) *x* **div** 5 = *x* mod 5;
b) 20 **div** *x* = 20 **mod** *x*;
c)* *x* **div** 5 = 8;
d)* 50 **div** *x* = 7.

Второй урок
ПЕРЕМЕННЫЕ И ПРЕДЛОЖЕНИЯ

В программе обычно используется много данных. Чтобы можно было указать, с какими данными требуется выполнить действия, данные обозначаются именами.

Имя составляется из букв и цифр, но оно обязательно должно начинаться буквой и посреди имени не должно быть промежутков. В отдельных случаях имя может быть составлено из одной буквы. Приведем примеры имен: *a*, *x*1, *x*2, *pi*, *сумма*.

Обозначенное именем данное называется **переменной.** Программу легче понять, если имена подбираются так, чтобы в них отражался смысл обозначаемых данных. Например, сумму цифр желательно обозначить именем *сумма* либо *s*, показатель степени — *степ* и т. д.

При выполнении программы каждой переменной отводится поле в памяти. Следовательно, имя переменной можно считать именем поля. В полях записываются значения переменных — конкретные числа (либо другие данные). Эти значения при выполнении программы могут меняться.

Запись значений в полях называется **присваиванием** и обозначается символом присваивания := (этот символ не следует путать с похожим на него символом равенства =). Слева от символа := записывается имя той переменной, которой нужно присвоить новое значение, справа — значение, которое следует присвоить указанной переменной. Например, *a* := 3. Здесь справа записано просто число. Однако чаще всего записывается выражение, значение которого требуется вычислить и присвоить переменной, расположенной слева от символа присваивания. Например, *s* := *x* + *y*.

В данном случае требуется вставить значения переменных *x* и *y* в выражение *x* + *y*, вычислить значение этого выражения и полученный результат присвоить переменной *s*. Итак, если переменной *x* ранее было присвоено значение 4, переменной *y* — значение 5, то переменной *s* компьютер присвоит значение 9.

Действия, которые должен выполнить компьютер, указываются в виде предложений. Предложения бывают разных видов в зависимости от того, какое действие требуется передать. Присваивание значения переменной осуществляется **предложением присваивания.** Итак, записанные выше предложения *a* := 3 и *s* := x + y являются предложениями присваивания.

Если переменной присваивается новое значение, прежнее уничтожается. Например, при выполнении предложений

k := 3;
k := *k* + 2;
k := 2 * *k*

значение переменной *k* будет равно 10. Первым предложением переменной *k* присвоено значение 3; после выполнения второго предложения значение *k* увеличилось на 2, т. е. стало равно 5; при выполнении третьего предложения

прежнее значение *k* удваивается и становится равным 10.

Записанные предложения образуют последовательность предложений. **Одно предложение от другого отделяется с помощью точки с запятой.** Предложения выполняются поочередно, так как они записаны в последовательности. Например, при выполнении несколько измененной последовательности предложений

$k := 3;$
$k := k * 2;$
$k := k + 2$

значение переменной *k* будет равняться 8.

Пример 1. Запишем предложение присваивания для вычисления значения переменной *y* по формуле $y = a^2 + b^3$.

В программе можно применять лишь те операции и те предложения, которые имеются в языке. Так как в языке ПАСКАЛЬ операция возведения в степень отсутствует, она заменяется умножением числа на само себя:

$y := a * a + b * b * b$

Пример 2. Напишем последовательность предложений для перестановки местами значений переменных *x* и *y*:

$p := x;$
$x := y;$
$y := p.$

Дополнительная переменная *p* потребовалась для того, чтобы сохранить значение переменной *x* (при выполнении предложения $x := y$ значение переменной *x* становится равным значению *y*, а имевшееся значение переменной *x* исчезает).

Пример 3. Значение переменной trio — трехзначное натуральное число. Запишем последовательность предложений для нахождения суммы цифр этого числа.

В языке ПАСКАЛЬ операция для нахождения цифр числа отсутствует. Это нужно выразить арифметическими операциями.

Первую цифру трехзначного числа получим, разделив заданное число на 100:

*s*1 := trio **div** 100.

Вторую цифру получить сложнее. Сначала нужно отбросить первую цифру числа, затем оставшееся двухзначное число поделить на 10: *s*2 := trio **mod** 10 **div** 10 .

Можно сделать иначе — сначала отбросить третью цифру, затем — первую:

*s*2 := trio **div** 10 **mod** 10.

Третья цифра равна остатку, полученному при делении числа на 10: *s*3 := trio **mod** 10.

Таким образом, нахождение суммы цифр числа trio можно записать такой последовательностью предложений:

*s*1 := trio **div** 100;
*s*2 := trio **div** 10 **mod** 10;
*s*3 := trio **mod** 10;
сумма := *s*1 + *s*2 + *s*3

Задачи для повторения

3. Даны значения переменных: $a = 1$, $b = 5$. Какими будут их значения после выполнения последовательности предложений?

$a := b;$
$b := a$

4. Какими значениями будут обладать переменные *x* и *y* после выполнения последовательностей предложений?

a) *x* := 15 **div** (8 mod 3);
   y := 17 **mod** *x* * 5 – 19 **mod** 5 * 2;
b) *x* := 2 * 5 **div** 3 **mod** 2;
   y := 2 * 5 **div** (3 **mod** 2);
   x := *x* * *y*;
   y := *y* * *y*.

5. Возведите выражения в степень, используя наименьшее количество операций сложения и умножения:

a)* x^7;
b) $(x + y)^8$;
c) x^9;
d) $(a + 1)^{21}$;
e) $(a + b + c)^{32}$;
f) $(a + b)^{100}$.

6. Значение переменной *a* — трехзначное натуральное число. Результат — двухзначное число *b*, которое получается из числа *a*, если вычеркнуть среднюю цифру. Напишите предложение (или последовательность предложений) для получения числа *b*.

Урок третий
УСЛОВНОЕ ПРЕДЛОЖЕНИЕ

Часто приходится выбирать один вариант решения (одно действие) из нескольких возможных. В этом уроке объясним, как записать выбор одного действия из двух возможных.

Разберем пример. Пусть значение переменной f зависит от значения переменной a : переменной f надо присвоить 1, если значение переменной a положительное; либо присвоить 0, если значение переменной a отрицательное или равно нулю. Таким образом должно выполняться одно из двух предложений: либо $f:=1$, либо $f:=0$; в зависимости от того, $a>0$ или $a\leq 0$. Математически это записывается так:

$$f=\begin{cases}1, \text{если } a>0,\\ 0, \text{если } a\leq 0.\end{cases}$$

Но как же такое действие указать компьютеру? Так как при составлении программы значение переменной a не известно (кроме того, оно может подбираться другое, а программа должна подходить та же), следует заранее предусмотреть все возможные случаи (варианты решения) и указать в программе, какие действия выполнять в каждом конкретном случае. Значит, необходимый вариант должен подобрать сам компьютер, решая задачу (т. е. выполняя программу), когда значение переменной a уже находится в его памяти. Сформулируем предписание, как нужно решать настоящую задачу: «Проверить, является ли $a>0$; если да, то выполнить предложение $f:=1$, в противном случае выполнить предложение $f:=0$».

В программировании принят более лаконичный способ записи выбора варианта — **условное предложение.** Действия указываются условными знаками или словами. В языке ПАСКАЛЬ условное предложение для выполнения вышерассмотренных действий записывается так:

```
if a>0
  then f:=1
  else f:=0.
```

Этим условным предложением указывается выполнение одного действия из двух: либо предложение $f:=1$, либо $f:=0$, в зависимости от значения a. Здесь применяются условные английские слова. Они переводятся так:

если $a>0$,
то $f:=1$
в противном случае (иначе) $f:=0$.

После слов **then** и **else** можно записать не только предложение присваивания, но и любое другое предложение, даже снова условное. Это будет показано в следующем уроке.

Условное предложение образует условие, следующее после слова **if,** и еще два предложения, следующие после слов **then** и **else.** Выполняется только одно из них. Если условие удовлетворено, то выполняется предложение, следующее после слова **then,** если не удовлетворено — предложение, следующее после слова **else.**

В условии для сравнения чисел употребляются следующие операции:

< меньше,
<= меньше или равно,
> больше,
>= больше или равно,
= равно,
<> не равно.

Сравниваются алгебраические значения чисел, т. е. учитываются и их знаки. Например, условие $1<2$ удовлетворено, а условие $-1<-2$ не удовлетворено.

Могут сравниваться значения не только переменных, но и выражений, например, $a+b$ **div** $2>=a$ **mod** 10. Обратим внимание, что операции сравнения выполняются в последнюю очередь — сначала выполняются все арифметические операции и лишь затем проводится сравнение.

Пример 1. Традиционная задача программирования — нахождение наименьшего (или наибольшего) значения из нескольких заданных значений переменных. Запишем условное предложение для присваивания переменной *min* наименьшего из двух значений переменных a и b. Когда значения переменных a и b равны, то переменной *min* можно присвоить любое значение. Запишем такое условное предложение:

```
if a<b
  then min:=a
  else min:=b.
```

Первую строку можно записать по-другому: **if** $a<=b$.

Когда значения a и b равны, в первом предложении переменной *min* присваивается значение b, а во втором — значение a. Это совершенно неважно, ведь $a=b$, поэтому обе записи дают один и тот же результат.

Пример 2. Запишем условное предложение для присваивания переменной k значения квадрата переменной a, если a — нечетное число, или значения куба — если a четное:

```
if a mod 2=1
   then k := a*a
   else k := a*a*a
```

Иногда действие нужно выполнить лишь в том случае, когда условие удовлетворено. В таком случае применяется сокращенное условное предложение (без слова **else**):

```
if...
  then...
```

Такое предложение называется упрощенным условным предложением.

Пример 3. Значение переменной a надо заменить ее модулем. Т. е. если $a<0$, то знак числа a надо поменять на противоположный — выполнить предложение $a:=-a$; когда $a>=0$, никакого действия выполнять не нужно. Это можно записать упрощенным условным предложением.

```
if a<0
   then a :=-a
```

Пример 4. Условное предложение второго примера поменяем на два предложения: присваивания и упрощенного условного.

```
k :=a*a;
if a mod 2=0
  then k := k*a
```

Здесь предложение $k:=k*a$ выполняется лишь в том случае, когда число a четное — тогда значение переменной k (а оно было бы равно квадрату числа a) еще раз умножается на a — получается куб.

Как видите, для одной и той же задачи можно составить разные программы.

Урок четвертый
УСЛОВНОЕ ПРЕДЛОЖЕНИЕ (ПРОДОЛЖЕНИЕ)

На прошлом уроке ознакомились с тем, как выбрать один вариант решения из двух. Однако часто приходится выбирать один из многих. Такой выбор можно осуществить несколькими условными предложениями, вложенными одно в другое.

Пример 1. Запишем предложение для вычисления значения y в зависимости от значения переменной x по следующей формуле:

$$y=\begin{cases}0, \text{ если } x<0,\\ x, \text{ если } 0\leq x<5,\\ 2x, \text{ если } x\geq 5.\end{cases}$$

Значение результата можно получить, если выполним одно из предложений присваивания : либо $y:=0$, либо $y:=x$, либо $y:=2*x$. Значит, существуют три варианта, зависимые от значения переменной x. Сначала сгруппируем их на два крупных варианта: когда $x<0$ и когда $x\geq 0$. Тогда запись условного предложения можно начать так:

```
if x<0
   then y :=0
   else ...
```

Далее разделим второй укрупненный вариант на два, вместо многоточия подставляя новое условное предложение:

```
if x<0
   then y :=0
   else if x<5
          then y :=x
          else y := 2*x
```

Задача решена.

Находящееся после слова **else** условное предложение выполняется лишь тогда, когда $x\geq 0$, поэтому имеющееся в нем условие по сравнению с соответствующей в формуле упрощено — достаточно проверить условие $x<5$.

Рассмотренный пример образует одно большое условное предложение, в которое после слова **else** вложено другое условное предложение. Условное предложение можно вставить не только после слова **else**, но и после слова **then**.

Пример 2. Значения переменных a, b, c — целые числа. Требуется определить наибольшее из них. Решение этой задачи осуществляется следующим условным предложением:

```
if a>b
   then if a>c
          then max :=a
```

```
        else max :=c
  else if b>c
        then max :=b
        else max :=c
```

Если $a>b$, то наибольшее число ищется среди *a* и *c*, в противном случае — среди *b* и *c*.

Те же самые действия можно записать и несколькими более простыми предложениями:

```
if a>b              или max :=a;
  then max :=a       if b>max
  else max :=b;        then max :=b;
if c>max             if c>max
  then max :=c         then max :=c.
```

Пример 3. Високосный год состоит из 366 дней, обычный — из 365. Год начала столетия будет високосным в том случае, если число сотен делится на 4 (например, 1600 год — високосный, а 1700 — невисокосный). Запишем предложение для определения количества дней в году, *m*:

```
if m mod 100=0
  then if m mod 400=0
         then m :=366
         else m :=365
  else if m mod 4=0
         then m :=366
         else m :=365
```

Вложенные условные предложения могут образовывать достаточно сложные структуры. Чтобы программу было легче читать, ее текст надо размещать как можно нагляднее. Как правило, слово **else** сравнивается с соответствующим ему словом **then.** Однако если после слова **else** следует длинное предложение (например, другое условное), то лучше слово **else** сравнивать с соответствующим словом **if.**

Чтобы правильно прочитать сложное условное предложение, надо найти в нем тройки слов **if, then** и **else** (или пары слов **if** и **then,** если предложение упрощенное), входящее в то же условное предложение. Когда в условном предложении имеются упрощенные условные предложения, может быть не ясно, какие предложения сокращены. Чтобы правильно понять такое условное предложение, надо представить себе, что все недостающие слова **else** записаны в конце предложения.

Задачи для повторения

7. Каково значение переменной *x* после выполнения следующих последовательностей предложений?

a)
```
x :=5;
if x>=5
  then x :=x*2
  else if x<=10
         then x :=-x;
x :=x*5
```

b)
```
x :=5;
if x>=5
  then x :=x*2
  else if x<=10
         then x :=-x
         else x :=x*5.
```

8. Какими значениями будут обладать переменные *a* и *b* после выполнения последовательности предложений?

```
a :=1; b :=2;
if a<b
  then a :=a+1
  else if a=b
         then a :=a+2
         else if a>b
                then b :=b+2.
```

9. Напишите условное предложение для определения значения переменной *f* по следующей формуле (*x* — целое число):

$$y=\begin{cases} -x^2, \text{ если } x<0, \\ 0, \text{ если } x=0, \\ x, \text{ если } x>0 \text{ и четное}, \\ x+1, \text{ если } x>0 \text{ и нечетное}. \end{cases}$$

10. Первые по счету Олимпийские игры состоялись в 1896 г. и проводятся через каждые четыре года. Если игры в тот год не состоялись, то он все равно считается олимпийским. Играм назначается номер по порядку. Напишите последовательность предложений для нахождения номера Олимпийских игр года *m* (если год *m* не олимпийский, номер игр считайте равным нулю).

Урок пятый
СОСТАВНОЕ ПРЕДЛОЖЕНИЕ

По правилам языка ПАСКАЛЬ после слова **then** или **else** можно писать только одно предложение. Если требу-

ется написать несколько предложений, они объединяются в одно составное. В начале составного предложения записывается служебное слово **begin** (начало), а в конце — **end** (конец).

Пример. Если $a<0$, то значения обеих переменных b и c увеличиваются на единицу.

```
if a<0
  then begin
         b:=b+1;
         c:=c+1
       end
```

Слова **begin** и **end** напоминают обычные скобки в арифметических выражениях. Таким образом, последовательность любых предложений можно сделать одним составным предложением, ограничивая его скобками предложений **begin** и **end.** Чтобы лучше выявить границы составного предложения, слова **begin** и **end** размещаются одно над другим. В составное предложение могут входить различные другие предложения, в том числе условные.

Пример 1. Значение переменных *h*1 и *min*1 — показываемое часами время (в часах и минутах). Запишем последовательность предложений для определения времени (в часах и минутах), спустя одну минуту.

```
min2:=min1+1;
if min2=60
  then begin
         min2:=0;
         h2:=h1+1;
         if h2=24
         then h2=0
       end
  else h2
```

Эта последовательность предложений выражает решение задачи без всяких хитростей. Можно составить и другие программы. Приводим еще один вариант решения:

```
min2:=min1+1;
h2:=(h1+min2 div 60) mod 24;
min2:=min2 mod60.
```

Пример 2. Значения переменных *a, b* и *c* — целые числа. Напишем последовательность предложений для определения максимального и минимального значений этих переменных.

```
if a>b
  then begin
         max:=a;
         min:=b
       end
  else begin
         max:=b;
         min:=a
       end
if c>max
  then max:=c;
if c<min
  then min:=c
```

Первым, более длинным условным предложением находится максимальное и минимальное значение при наличии только двух переменных. Вторым и третьим условными предложениями проводится проверка, не является ли значение переменной *c* больше или меньше уже найденного максимума или минимума. Если да, то выполняются соответствующие изменения.

Задачи для повторения

11. Читатель начал решать контрольные задачи данного урока программирования, когда электронные часы показывали *h*1 часов и *min*1 минут, а кончил, когда было *h*2 часов и *min* 2 мин. Составьте последовательность предложений, с помощью которой определяется, сколько времени (часов и минут) читатель решал эти задачи (примем, что задачи решались не дольше суток).

Урок шестой
ЦИКЛ WHILE

Для решения рассматривавшихся до сих пор задач компьютер вряд ли был нужен. Теперь ознакомимся с такими задачами, которые без компьютера решить было бы весьма трудно. Так как в них одно или несколько предложений нужно повторить много раз. Эти повторяемые части программы указываются предложениями цикла (часто говорится короче — циклами). Как выполняются такие предложения, будет ясно после анализа нескольких простых примеров.

Пример 1. Напишем предложения для нахождения первой цифры заданного натурального числа *n*.

Так как неизвестно, сколько цифр содержится в числе, то для нахожде-

ния первой цифры надо данное число делить на 10 и повторять это действие до тех пор, пока число станет однозначным (оно и будет результатом). Это повторение выражается предложением цикла:

```
p:=n;
while p>9 do
    p:=p div 10
```

Предложение цикла начинается словом **while** — пока. Дальше (после слова **do** — выполнить) следует предложение, которое повторяется до тех пор, пока удовлетворено условие, указанное в заголовке.

Как выполняется цикл?

Прежде всего проверяется условие. Если оно не удовлетворено, то предложение, идущее после слова **do,** не выполняется ни разу (в приведенном примере так будет, когда *n* — однозначное число). Если условие удовлетворено, то выполняется идущее после заголовка предложение и вновь следует возвращение к условию. Действия повторяются, пока условие удовлетворено. Когда условие не удовлетворено, цикл завершается.

Проанализируем работу цикла, используя ранее приведенный пример, при конкретном значении переменной *n,* например, когда $n=325$. Сразу условие удовлетворено, ибо $325>9$. Поэтому выполняется предложение присваивания *p*:=*p* **div** 10. Теперь $p=32$. Второй раз проверяемое условие удовлетворено, ибо $32>9$. Повторно выполняется предложение *p*:=*p* **div** 10. Теперь $p=3$. Условие не удовлетворено, так как $3<9$. Поэтому предложение присваивания не выполняется, и цикл заканчивает свою работу. Следовательно, значение переменной *p* будет результатом — первой цифрой данного числа.

Заголовок цикла управляет повторением лишь одного, следующего за ним предложения. Если надо повторить несколько предложений, то они объединяются в составное предложение (ограничивается словами **begin** и **end**).

Пример 2. Напишем последовательность предложений для нахождения суммы квадратов чисел от 1 до 100:

```
s:=0; i:=1;
while i<=100 do
begin
    s:=s+i*i;
    i:=i+1
end
```

Сумма в цикле возрастает с прибавлением к ней квадратов значения переменной *i*. Каковы значения этой переменной? При первом суммировании значение *i* остается таким, каким было дано перед циклом, т. е. 1. При повторении цикла значение *i* возрастает на единицу. Таким образом, при втором суммировании значение *i* будет равно 2, при третьем — 3 и т. д. После выполнения цикла в сотый раз значение переменной *i* будет равно 101, условие уже не будет удовлетворяться, и действия цикла будут закончены.

Обратим внимание на одну важную деталь. Всякий раз при выполнении цикла квадраты значения переменной *i* прибавляются к бывшему значению переменной *s*. Значит, переменная *s* еще перед циклом должна приобрести какое-нибудь значение, иначе при выполнении цикла в первый раз мы не смогли бы начать суммирование — переменная *s* не имела бы никакого значения (в таких случаях говорят, что переменная не определена). Поэтому в представленном примере перед предложением цикла переменной *s* (будущей сумме) присваивается 0.

При составлении циклов нетрудно ошибиться и записать не те действия, которые имелись в виду. Поэтому важно хорошо проверить программу, найти и исправить допущенные в ней ошибки. Одной из наиболее часто встречающихся ошибок является никогда не кончающийся цикл (называемый вечным).

Пример 3. Программист, составляя фрагмент программы суммирования квадратов чисел, забыл увеличить значение переменной *i* на единицу:

```
s:=0; i:=1;
while i<=100 do
    s:=s+i*i
```

Значение переменной *i* в цикле не изменяется. Следовательно, оно все время будет равно единице. Поэтому условие, находящееся в заголовке цикла, всегда будет удовлетворено, и компьютер никогда не сможет закончить цикл.

Встречаются и другие ошибки. Многие из них можем найти сами, проверяя программу. Для этого подбираем менее сложные исходные значения переменных и выполняем записанные в программе действия так, как бы выполнял их компьютер. На листке бумаги записываем используемые в программе переменные и их исходные значения. Когда переменной присваивается новое значение, то прежнее зачеркиваем, а на его место пишем новое. Если таким образом полученные результаты не совпадают с заданными — значит, в программе допущена ошибка.

Пример 4. Для нахождения произведений *s* всех цифр заданного числа *n*, за исключением нулей, написан следующий фрагмент программы:

```
p:=n; s:=0;
while p>0 do
  begin
    if p mod 10<>0
      then s:=s*(p mod 10);
    p:=p div 10
  end
```

Когда *n*=23, ясно, что результат должен быть 6. Проверим, таким ли будет результат, когда выполним представленную программу. Значения переменных перед циклом суть *p*=23, *s*=0. Условие в заголовке цикла удовлетворено. Выполнив цикл, получим *p*=2, *s*=0. Еще раз выполнив цикл, получаем *p*=0, *s*=0. Теперь условие не удовлетворено, поэтому цикл не выполняется. Полученное значение произведения не равно 6. Значит, есть ошибка. Разобрав программу, можно заметить, что вместо предложения *s*:=0 должно быть предложение *s*:=1.

Для проверки подобранные начальные значения переменных называются контрольными данными. Они должны быть подходящими и свойственными для данной программы. Например, если в программе много условных предложений, контрольные данные надо подобрать так, чтобы хоть по одному разу выполнялась каждая часть условного предложения. При проверке программ, в которых употребляются циклы, стоило бы подобрать такие контрольные данные, чтобы цикл не выполнялся ни разу, выполнялся один раз, несколько раз.

Задачи для повторения

12. Следующая последовательность предложений предназначена для определения количества цифр в заданном натуральном числе *n*:

```
a:=n; сколько:=0;
while a>=0 do
  begin
    сколько:=сколько+1
    a:=a div 10
  end.
```

Однако в последовательности содержатся ошибки. Исправьте их.

13. Значение переменной *n* — натуральное число. Для нахождения числа *a*, цифры которого были бы расположены в обратном порядке, чем у данного числа *n* (например, если *n*=351, то *a*=153), был составлен следующий фрагмент программы:

```
p=n;
while p>=0 do
  begin
    a:=a+p mod 10
    p:=p div 10
  end
```

Однако в программе содержатся ошибки. Исправьте их.

Урок седьмой
ЦИКЛ FOR

В программах довольно часто встречаются циклы, имеющие в заголовке переменную, значение которой увеличивается на единицу при каждом повторении цикла: при этом цикл повторяется до тех пор, пока значение этой переменной не достигает определенного предела. Для такого случая в языке ПАСКАЛЬ существует еще один вид цикла.

Пример 1. Напишем предложения суммирования квадратов чисел от 1 до 100, используя цикл нового вида:

```
s:=0;
for i:=1 to 100 do
  s:=s+i*i
```

Заголовок цикла начинается словом **for** (для), после которого следует имя переменной цикла и указывается, для каких значений этой переменной должен выполняться цикл. При выполнении цикла сначала его переменной присваивается указанное в заголовке

значение (в данном примере i:=1) и выполняется предложение, находящееся после слова **do** (выполнить). Затем значение переменной цикла увеличивается на единицу и снова выполняется предложение, находящееся после слова **do**, и т. д. Когда переменная цикла приобретает значение, указанное после слова **to** (до), цикл выполняется в последний раз. Значит, записанный здесь цикл будет выполняться 100 раз (значение переменной цикла будут i=1, 2, 3, ... 99, 100).

Значения переменной цикла увеличиваются на единицу автоматически, поэтому менять их внутри цикла нельзя.

Начальное и конечное значения переменной цикла могут указываться не только целыми числами, но и выражениями или переменными.

Пример 2.

```
s:=0;
for i:=m to n do
    s:=s+i*i
```

Здесь вычисляется сумма квадратов целых чисел от m до n. Придавая переменным m и n разные значения (целые числа), вычислим сумму квадратов целых чисел в разных интервалах. В частном случае, когда m=1 и n=100, результат получается такой же, как в предыдущем примере. Таким образом, этот фрагмент программы более универсален.

Цикл выполняется $n-m+1$ раз (например, когда $m=-5$, $n=0$, шесть раз).

Если начальное значение переменной цикла превосходит конечное, то цикл не выполняется ни разу.

Пример 3. Запишем предложения для нахождения факториала натурального числа n.

```
f:=1;
for k:=1 to n do
    f:=f*k
```

Цикл выполняется n раз, т. е. столько раз, сколько имеет номинал числа, факториал которого вычисляется. Каждый раз при выполнении цикла результату f присваивается значение, которое равно предыдущему значению f, умноженному на значение переменной k.

Подставленный цикл состоит только из одного предложения, находящегося после заголовка. Когда нужно повторить несколько предложений, их следует объединить в одно составное предложение.

Пример 4. Для нахождения суммы

$1 \cdot 1!+2 \cdot 2!+...+n \cdot n!$

запишем следующие предложения:

```
s:=0; f:=1;
for i:=1 to n do
    begin
        f:=f*i;
        s:=s+i*f
    end
```

В цикл может войти другое предложение цикла или условное предложение. Нет никаких ограничений, чтобы выполнить предложение цикла в составном или условном предложении.

Пример 5. Дано натуральное число n. Запишем предложения для нахождения наибольшего его делителя, не равного самому числу n:

```
for i:=1 to n-1 do
    if n mod i=0
      then делитель := i
```

Этот фрагмент программы можно усовершенствовать, уменьшая количество повторений цикла: достаточно проверить интервал от 1 до n **div** 2 (убедитесь!). Кроме того, делитель четных чисел можно найти сразу, без цикла (тогда в начале потребовалось бы условное предложение: если число четное, делитель равен просто числу n **div** 2, если нечетное — делитель определяется при помощи цикла).

Когда какой цикл применять? Если количество повторений известно заранее, до выполнения цикла, то лучше подходит цикл **for**, поскольку он записывается короче и нагляднее. Если количество повторений до выполнения цикла установить нельзя, следует применять цикл **while**.

Задачи для повторения

14. Напишите последовательность предложений для нахождения суммы

$1+11+111+...+111...1,$

если последнее слагаемое состоит из n цифр?

15. Напишите последовательность предложений для вычисления суммы (n — любое нечетное натуральное число):

$1.3+3 \cdot 5+...+n(n+2).$

Урок восьмой
ЦИКЛЫ В ЦИКЛЕ

Как уже говорили, в цикл может входить другой цикл, а в этом внутреннем цикле могут находиться новые циклы и т. д.

Пример 1. Запишем предложения для нахождения суммы

$$1^k+2^k+...+n^k.$$

```
сумма :=0;
for j:=1 to n do
    begin
      p:=1;
      for j:=1 to k do
     p:p*i;
     сумма := сумма+p
   end
```

Действия внешнего цикла (его заголовок **for** *i*:=1 **to** *n* **do**) выполняются *n* раз. В этом цикле имеется составное предложение. В нем находится другой, внутренний цикл для возведения члена *i* в степень *k*. Как выполняются эти циклы? Сначала значение переменной *k* внешнего цикла становится равным единице, тогда выполняется внутренний цикл, действия которого повторяются *k* раз (значение переменной *j* меняется от 1 до *k*). Затем снова возвращаемся к внешнему циклу, значение переменной *i* увеличено на единицу (т. е. значение *i* становится равным 2), и снова *k* раз выполняются действия внутреннего цикла. Затем снова возвращаемся к внешнему циклу и т. д. Это повторяется до тех пор, пока значение переменной *i* внешнего цикла становится равным *n*. Например, если *n*=20, *k*=3, действия внешнего цикла выполняются 20 раз, а внутреннего 20·3=60 раз.

Убедитесь, для того чтобы найти сумму $1^1+2^2+...+n^n$, достаточно в рассмотренной программе изменить лишь заголовок внутреннего цикла — он должен быть следующим: **for** *j*:=1**to** *i* **do**

Пример 2. Требуется найти все двухзначные числа, которые делятся на сумму своих цифр.

Задачу можно решить двумя способами: 1) найти цифры каждого двухзначного числа и проверить, делится ли оно на сумму этих цифр; 2) взять все возможные пары цифр, составить из них двухзначное число и проверить, делится ли это число на сумму цифр.

Задачу решим вторым способом.

```
сколько :=0;
for a:=1 to 9 do
    for b:=0 to 9 do
    begin
          dd:=a*10+b;
          s:=a+b;
          if dd mod s=0
            then сколько := сколько+1
    end
```

Двухзначное число обозначили именем *dd*, его цифры — именами *a* и *b*, сумму цифр — *s*, а результат — именем *сколько*. Внешним циклом перебираются все цифры от 1 до 9 — это первая цифра двухзначного числа. Внутренним циклом перебираются все цифры от 0 до 9 (вторая цифра двухзначного числа может быть равна нулю). Таким образом внешний цикл выполняется 9 раз, а внутренний 9×10 =90 раз.

Мы уже рассмотрели предложения четырех видов: присваивания, составное, условное и цикла. Предложение присваивания элементарное, с его помощью переменной присваивается значение. Составное предложение, условное предложение и цикл называются структурами управления. Ими выражаются предписания управления порядком выполнения других предложений. Все рассмотренные структуры управления равноправны: они могут входить одна в другую без каких-либо ограничений. Рассмотренных структур управления достаточно для составления программ довольно сложных задач.

Задачи для повторения

16. Сколько раз выполняются внешний и внутренний циклы в фрагментах следующих программ?

a)*
```
a:=0;
while a<=5 do
while a:=a+1
```

b)
```
a:=0;
while a<=10 do
  while a<=20 do
    a:=a+1
```

c)
```
a:=0;
while a<=20 do
  while a<=10 do
    a:=a+1
```

17. Запишите последовательность предложений для нахождения всех трехзначных натуральных чисел, средняя цифра которых равна сумме первой и второй цифр.

Урок девятый
ВВОД И ВЫВОД

Компьютер выполняет действия с данными. Перед выполнением программы имеем исходные данные. После выполнения программ получаем результаты или конечные данные.

До сих пор, составляя программы, мы не интересовались ни тем, как исходные данные попадают в память компьютера, ни тем, как из нее забрать результат. Говорится, что исходные данные надо ввести в память компьютера, а результаты — вывести из нее.

Наиболее широко используется ввод значений исходных данных непосредственно с клавиатуры компьютера. Результаты печатаются на экране дисплея или на бумаге.

В программе ввод и вывод данных указываются предложениями ввода и вывода. Предложение ввода начинается словом *read* (читать). После него в скобках записываются имена тех переменных, которым должны быть присвоены значения введенных данных.

Пример 1. Напишем последовательность предложений для ввода и сложения двух чисел:

```
read (a);
read (b);
s:=a+b
```

Компьютеру следует представить данные в том порядке, как они записаны в предложениях ввода. В данном примере прежде всего набирается значение переменной *a*, а затем значение переменной *b*.

Предложения ввода можно объединить в одно предложение и с его помощью ввести несколько данных. В том случае переменные в предложении ввода отделяются друг от друга запятой. Например, вместо предшествующих двух предложений ввода можно написать одно:

```
read (a, b);
s:=a+b.
```

Предложение вывода (печатания) начинается словом *write* (писать). После него в скобках указываются те данные, которые надо печатать.

Данные в предложении вывода отделяются друг от друга запятой.

Пример 2. Компьютер должен прочитать два исходных данных и напечатать их в обратном порядке, т. е. сначала должен напечатать второе прочитанное число, затем — первое. Это можно указать такими предложениями.

```
read (a, b);
write (b, a)
```

Пример 3. Исходные данные — 100 чисел. Нужно найти и напечатать их сумму.

Эта задача похожа на пример 1. Однако исходных данных здесь много, и потребовалось бы использовать 100 переменных, а это не очень удобно. Поэтому будем решать другим способом.

```
s:=0;
for k:=1 to 100 do
    begin
      read (a);
      s:=s+a
    end
write (s).
```

Цикл повторяется 100 раз. В цикл вводится исходное данное и добавляется к сумме *s*. При повторении цикла новое исходное данное вводится на место предшествующего, ибо предшествующее данное уже сложено с суммой и больше не нужно. Поэтому ту же переменную (а тем самым и то же самое место в памяти компьютера) можно использовать для хранения других значений исходных данных.

Перед выполнением цикла сумме (переменной *s*) присваивается нулевое значение. Такое действие необходимо, так как при выполнении цикла в первый раз мы не смогли бы начать суммирование — не к чему было бы прибавлять, т. е. значение *s* было не определено.

Пример 4. Исходные данные — последовательность чисел, не равных нулю. Сколько чисел в последовательности — заранее не известно. Однако известно, что последовательность кончается, когда встречается число нуль. Иначе говоря, признак конца последовательности — нуль. Запишем фрагмент программы для нахождения и

печатания суммы всех чисел этой последовательности:

```
s:=0;
read (a);
whil a<> 0 do
  begin
    s:=s+a;
    read (a)
  end;
write (s)
```

Выполнение цикла повторяется до тех пор, пока $a \neq 0$. Заметим, что последний элемент вводимой последовательности — нуль — будет тоже прибавлен к общей сумме *s*.

Эта программа более универсальна, чем программа примера 3, она годится для суммирования любого числа исходных данных. Однако она не годится для суммирования таких данных, в которых встречаются нули.

В предложении вывода можно писать не только имена переменных, но и числа или выражения. В таких случаях печатаются те же числа или значения выражений. Например, после выполнения предложений

```
mn:=11;
d:15;
write (1985, mn, d+5)
```

компьютер напишет три числа:

1985 11 20

Пример 5. Напишите последовательность предложений для нахождения квадратов и кубов всех целых чисел из интервала [*m*; *n*]. Каждый раз должно быть напечатано число, его квадрат и куб.

```
read (m, n);
for k:=m to n do
  write (k, k*k, k*k*k)
```

Например, если исходные данные — 2 и 5, то компьютер напечатает следующие числа:

2 4 8 3 9 27 4 16 64 5 25 125.

Задачи для повторения

18. После выполнения последовательности предложений компьютер напечатал число 2. Какое было значение исходного данного?

```
read (a);
s:=0;
for i:=a to 5 do
  if i>0 then s:=s-1
         else s:=s+1;
write (s)
```

Урок десятый
ПЕЧАТАНИЕ РЕЗУЛЬТАТОВ

Напечатать можно не только числа, но и текст. Текст записывается в предложении вывода между апострофами. Например, после выполнения предложения

write ('***** год рождения 1954 *****')

компьютер напечатал такой же самый текст, только без апострофов:

**** ГОД РОЖДЕНИЯ 1954 *****

Обратим внимание на то, что пробел тоже является символом. Компьютер во время печатания оставляет столько пробелов, сколько их было между апострофами.

Обратим внимание и на то, что некоторые компьютеры печатают только прописными буквами. Мы можем записать программу так, как это удобнее и нагляднее. На языке ПАСКАЛЬ программы принято записывать строчными буквами, а печатаемый текст — прописными. Компьютер печатает данные на листе бумаги друг за другом в одну строку, пока не заполняет ее целиком. Новая строка начинается автоматически только при нехватке на текущей строке места для печатаемого данного. Если нужно начать печать с новой строки, употребляется предложение *writeln* (оно произошло от слов *write line* — писать с новой строки). Например, после выполнения предложений:

write ('ИНФОРМАТИКА');

write ('И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА') компьютер напечатает все на одной строке:

ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА, а после выполнения предложений:

write ('ИНФОРМАТИКА');
writeln
write ('И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА') — на двух строчках.

ИНФОРМАТИКА
И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА.

В предложении *writeln* можно указать и данные, предназначенные для печати. В таком случае сначала печатаются данные, а потом выполняется переход на новую строку. Таким обра-

зом, две ранее отпечатанные строки можно получить и следующими двумя предложениями:

```
writeln ('ИНФОРМАТИКА');
write ('И ВЫЧИСЛИТЕЛЬНАЯ
        ТЕХНИКА').
```

Пример 1. Исходные данные — последовательность чисел, не равных нулю. Признак конца последовательности — нуль. Напишите последовательность предложений для печатания каждого числа с новой строки, а рядом с этим числом — слова ДА, если это число делится на 11, или слово НЕТ — если не делится.

Для этой задачи составлен такой фрагмент программы:

```
read (s);
while s < > 0 do
  begin
    write (s); write (' ');
    if s mod 11=0
      then writeln ('ДА')
      else writeln ('НЕТ');
    read (s)
  end
```

Предложением *write* (' ') указали, что после числа перед словом ДА или НЕТ был бы оставлен пробел (между числами пробелы оставляются автоматически, а текст печатается без пробелов).

Укажем и иначе записанную последовательность предложений для решения этой задачи:

```
read (s);
while s <> 0 do
  begin
    if s mod 11=0
      then writeln (s, ' ДА')
      else writeln (s, ' НЕТ');
    read (s)
  end
```

Пример 2. Запишем фрагмент программы для напечатания прямоугольника, составленного из *m*n* звездочек (*m, n* — исходные данные, натуральные числа). Например, если исходными данными являются числа 3 и 5, надо напечатать

```
*****
*****
*****
```

Приведем последовательность предложений для напечатания этого прямоугольника:

```
read (n, m);
for h:=1 to n do
  begin
    for i:=1 to m do
      write ('*');
    writeln
  end
```

В программе два цикла: внешний подсчитывает строчки звездочек, образующих прямоугольник, внутренний — количество звездочек в строчках. Каждый раз после напечатания *m* звездочек осуществляется переход на новую строку.

Пример 3. Исходные данные — последовательность натуральных чисел. Признак конца последовательности — нуль. Составим фрагмент программы для напечатания чисел этой последовательности, располагая их по три числа в строку (тремя столбиками). Например, дана последовательность.

7 8 4 15 20 3 8 7 9 25 50 0,

то надо печатать:

```
 7  8 4
15 20 3
 8  7 9
25 50
```

Как указать в программе расположение чисел по три в строку? Прежде всего так: пронумеровать и напечатать члены последовательности — когда номер напечатанного члена последовательности делится на три, должен осуществляться переход на новую строку. Приводим таким способом составленную программу:

```
ном:=0;
read (x);
while x < > 0 do
  begin
    ном:=ном+1;
    write (x);
    if ном mod 3=0 then
    writeln
    read (x)
  end
```

Номер элемента последовательности определяется переменной *ном*. После напечатания члена, номер которого делится на 3, осуществляется переход на новую строку.

Задачи для повторения

19. Дана последовательность предложений:

```
read (n);
for i:=1 to n do
    begin
      for j:=1 to n do
          if i>=j then write ('A')
                  else write ('B');
      writeln
end
```

Что напечатает компьютер после выполнения этой последовательности, если исходные данные: a) 4; b) 7?

20. Дана последовательность предложений:

```
read (n, m);
for a:=1 to n do
    begin
      write ('U')
      for b:=2 to m-1 do
         if a=n
           then write ('U');
           else write ('=');
    writeln ('U')
end
```

Что напечатает компьютер после выполнения последовательности, если исходные данные числа 4 и 5?

УРОК ОДИННАДЦАТЫЙ
Структура программы

Рассмотренные в предыдущих уроках последовательности предложений мы иногда называли программами, хотя в действительности это еще не были законченные программы. Однако их без труда можно дополнить до программ, понятных компьютеру.

Программу, записанную на языке ПАСКАЛЬ, образуют следующие части: 1) заголовок программы; 2) описания; 3) составное предложение; 4) символ конца программы — точка.

Пример 1. Составим программу для нахождения и печатания суммы двух чисел:

```
program сумма (input, output);
   var a, b, s:integer;
begin
  read (a, b);
  s:=a+b;
  write (s)
end.
```

Первая строка этого примера — заголовок программы. Он всегда начинается словом *program.* Затем записывается имя программы, которое подбирает программист. Мы эту программу назвали именем *сумма.* Могли назвать и иначе. Однако надо постараться, чтобы имя соответствовало смыслу задачи.

В скобках после имени записывается слово *input* (ввод), если в программе употребляются предложения для введения данных, и слово *output* (вывод), если употребляются предложения для печатания данных. (Для некоторых компьютеров скобки и между ними имеющуюся информацию можно пропускать. Так мы и будем поступать в дальнейшем.)

Вторая строка программы — описание переменных. Она начинается словом **var** (произошло от слова variable — переменная), после которого перечисляются имена всех используемых в программе переменных (они отделяются друг от друга запятой). Слово *integer* (целое) говорит о том, что в ходе выполнения программы переменные *a*, *b* и *s* могут принимать только значения целых чисел. Абсолютную величину этих значений ограничивает конструкция компьютера.

Все употребляемые в программе имена переменных должны быть описаны.

После описания следует составное предложение. Таким образом, можно сказать, что все действия программы записываются одним составным предложением. Разумеется, это предложение может быть очень сложным и для его записи может понадобиться множество страниц.

Обратим внимание на знаки препинания. После заголовка и после описаний ставится точка с запятой, а в конце всей программы — точка. Предложения отделяются друг от друга точкой с запятой.

Пример 2. Составим программу для нахождения всех делителей данного натурального числа *n*.

Простейший способ решения этой задачи — проверить делимость числа *n* на все подряд следующие числа 1, 2, 3, ...,*n*.

Составляем программу:

```
program делители;
```

```
    var n, d : integer;
begin
    read (n);
    for d:=1 to n do
        if n mod d=0
            then write (d)
end.
```

Например, если исходное данное 30, компьютер должен напечатать:

1 2 3 5 6 10 15 30

Эту программу мы еще можем усовершенствовать — сделать эффективней, т. е. выполняющей меньше действий. Легко убедиться, что в интервале [*n* **div** 2+1; *n*−1] делители числа *n* отсутствуют. Поэтому не следует указывать компьютеру, чтобы он их искал. Приводим вариант усовершенствованной программы:

```
program делители;
    var n, d: integer;
begin
    read (n);
    for d:=1 to n div 2 do
        if n mod d=0
            then write (d);
    write (n);
end.
```

Текст программы, как и исходные данные, записывается в память компьютера. При выполнении программ компьютер печатает листинг. Это — текст программы, различные заголовки, объяснения и результаты расчетов.

Задачи для повторения

21. Автоморфным называем такое число, цифры которого совпадают с последними цифрами его квадрата. Например, $6^2=36$, $25^2=625$, $76^2=5776$.

Для нахождения автоморфных чисел в интервале [*m*; *n*] составлена такая программа:

```
program автоморфные;
    var m, n, x, d: integer;
begin
    read (m, n);
    d:=10;
    for x:=m to n do
        begin
            while ...do
                d:=d*10;
            if ... then
            writeln (x, x*x)
        end
end.
```

На месте многоточий вставьте пропущенные условия.

22. Исходное данное — натуральное число *q*, выражающее площадь. Напишите программу для нахождения всех таких прямоугольников, площадь которых равна *q* и стороны выражены натуральными числами. Длину сторон каждого прямоугольника печатайте с новой строки. Например, если *q*=20, то результат должен быть следующим:

```
1    20
2    10
4     5
```

23. Составьте программу для графического изображения делимости чисел от 1 до *n* (*n* — исходное данное). В каждой строке нужно печатать число и столько плюсов, сколько делителей у этого числа. Например, если исходное данное 4, то должно быть напечатано:

```
1+
2++
3++
4+++
```

24. Дана программа:

```
program xxx;
    var a, b: integer;
begin
    read (a);
    b:=0;
    while a<>0 do
        begin
            b:=b*10+a mod 10;
            a:=a div 10
        end
    write (b)
end.
```

Что напечатает компьютер после выполнения этой программы, если *a*=13 305? Решение какой задачи выражает эта программа?

25. Дана программа:

```
program отгадай;
        var a, b, c, i, j: integer;
begin
    read (n);
    a:=1; b:=1;
    for i:=0 to n do
        begin
            for j:=1 to b do
                write ('*');
            writeln;
```

```
        c:=a+b;
        a:=b; b:=c
    end
end.
```

Что напечатает компьютер после выполнения программы, если исходное данное 6? Решение какой задачи выражает эта программа?

Урок двенадцатый
МЕТОДИКА ПРОГРАММИРОВАНИЯ

Рассмотренных конструкций программирования вполне достаточно для составления программ многих задач. Если задача сложная, составить для нее программу непросто. Поэтому при написании программ следует придерживаться определенной системы правил — методики программирования.

Программирование задачи можно разбить на следующие части:

1. Формулировка задачи.
2. Выбор или составление алгоритма решения.
3. Составление программы.
4. Проверка программы.
5. Усовершенствование программы.
6. Отладка программы.

Вкратце рассмотрим каждую из них.

Формулировка задачи. Перед программированием задачи надо подробно уяснить все требования к будущей программе — составить точное задание. В противном случае программист может не предусмотреть отдельные детали. В формулировке задачи требуется четко указать, какие исходные данные следует использовать, что должна выполнить программа и какие результаты должны быть получены. Здесь важно установить и форму вывода результатов: будут ли они изложены в таблице, изображены графически, оформлены заголовками или просто напечатаны в строку. Позднее при составлении программы определяется, какие действия надо выполнить для получения требуемого результата.

Выбор или составление алгоритма решения. Алгоритм решения задачи должен знать программист. Ведь сама программа и представляет собой точное описание решения задачи, которую одинаково понимают и человек, и компьютер. Компьютер, решая задачу, выполняет ту большую, рутинную вычислительную работу, которая записана в программе. Таким образом, программист прежде всего должен разобраться, как решается данная задача. При этом он может использовать известные методы или разработать собственные.

Составление программы. Программа каждой задачи своеобразна. При решении одной и той же задачи можно составить много разных программ для разных вариантов решения. Программирование — это творческий процесс, поэтому общих рецептов для составления программ не существует. Однако можно рекомендовать некоторые приемы, используя которые программист облегчит и ускорит свою работу. Одним из таких приемов является дробление задачи на части. Программировать большую задачу сразу трудно или вообще невозможно. Поэтому задача разбивается на несколько более мелких частей. Каждая часть решается (программируется) отдельно. Если отдельная часть еще слишком велика, то ее дробление следует продолжать до тех пор, пока вновь полученные части не будут достаточно простые и наглядные.

Проверка программы. При составлении программ очень легко ошибиться. Поэтому программу надо тщательно проверить еще до ее введения в компьютер.

При проверке программы следует убедиться: 1) нет ли синтаксических ошибок (какое-либо предложение записано неправильно, пропущен знак препинания и т. п.); 2) описаны ли все имена переменных; 3) определены ли значения всех переменных; 4) конечны ли действия программы; 5) даст ли программа правильные результаты.

Самое важное — правильность результатов. Один из простейших способов проверки — подобрать какие-либо исходные данные и самому программисту выполнить записанные в программе действия. Все действия надо выполнять так, как их выполнил бы компьютер, — побуквенно, механически, не углубляясь в их смысл. Тогда мы получим такие результаты, какие получил бы компьютер, а не такие,

какие мы хотим или надеемся получить.

Подобранные для проверки исходные данные называются контрольными. Они должны быть характерны для проверяемой программы, такими, чтобы проверялись все части программы (например, если в программе имеются условные предложения, надо подобрать такие контрольные данные, чтобы выполнялась каждая ветвь условного предложения). Кроме того, контрольные данные надо подобрать так, чтобы результат можно было бы без труда подсчитать самим.

Усовершенствование программы. Не сразу удается составить красивую и экономичную программу. Иногда программа уже составлена, а программисту приходят новые идеи для ее усовершенствования: сделать короче, яснее или экономичнее. Пример усовершенствования программы был рассмотрен при составлении программы для нахождения всех делителей данного числа (см. урок II).

Отладка программы. Программист, составив программу и записав ее в компьютер, обычно рассчитывает, что получит правильный результат. Однако даже в хорошо проверенной программе встречаются ошибки. Компьютер печатает сообщения обо всех обнаруженных им ошибках. Программист должен рассмотреть каждую ошибку и исправленную программу вновь ввести в компьютер. Этот процесс повторяется до тех пор, пока в программе не останется ошибок. Так исправляются все синтаксические ошибки.

Смысловые ошибки выявить труднее. Их ищут, вводя в компьютер исходные данные и сравнивая напечатанные



ОСИПОВ Л. А.

Первая персональная ЭВМ была создана в нашей стране в 1966 г., на 10 лет раньше, чем в США. Это была МИР (машина инженерных расчетов), разработанная под руководством академика В. М. Глушкова. МИР имела вид стола с пишущей машинкой. ЭВМ предназначалась для самостоятельной работы пользователя, который включал машину, решал свою задачу и выключал ее. Простота управления машиной обусловлена операционной системой, записанной в постоянной памяти.

В МИРе использовался язык программирования АЛМИР-65, являющийся русифицированным развитием АЛГОЛа-60. Язык позволял устанавливать практически любую разрядность вычислений оператором «РАЗР»р. В программе устранена необходимость описания типов величин (целые, вещественные). Тип величин определялся автоматически типом присваиваемого значения. В качестве операторных применялись обычные круглые скобки, что повышало наглядность программы.

В АЛМИРе введены операторы печати списка величин или выражений, таблиц и графиков. Вывод значений автоматически сопровождался именами величин или выражениями. Такие операторы вывода существенно упрощают программирование.

Весьма полезным на АЛМИРе явилось включение в состав операций языка вычисление математических функций, сумм, произведений и определенных интегралов. В структуре программы описательная часть с исходными данными, массивами и внутренними функциями отнесена в конец и отделена от операторной части служебным словом «где». Это позволяет простой заменой описательной части варьировать варианты расчетов.

Наиболее распространенная ошибка при программировании — численно неопределенная величина. Во многих языках на место неопределенной величины подставляется произвольное число и выдается результат, ошибочность которого можно установить только параллельным расчетом. С другой стороны, выявить неопределенную величину в сложной программе не так просто. На АЛМИРе при неопределенных величинах печатается алгебраическое выражение, из которого сразу выявляются такие величины.

Клавиатура МИРа построена с учетом психофизиологических факторов. В ней применен совмещенный русско-латинский алфавит, вынесенный на верхний регистр. При этом в 45 клавишах был расширен состав математических символов. Такое построение клавиатуры резко снизило путаницу в наборе букв и увеличило скорость ввода в ЭВМ программ и данных. В порядке дальнейшего развития в клавиатуру можно ввести переключатель «прописные — строчные буквы». В МИРе удобным было также устройство печати на обычную бумагу.

В 1969 г. появилась ЭВМ МИР-2. В ней впервые был применен

компьютером результаты с результатами, известными заранее. Годятся те же контрольные данные, которые применялись при ручном способе проверки правильности программы. Только теперь можно брать и более сложные данные.

Итак, мы убедились, что подготовка программы — длительный и кропотливый труд. Однако когда программа составлена и отлажена, ее можно выполнять много раз при вводе все новых исходных данных. Такой программой могут воспользоваться и другие.

Урок тринадцатый
КУЛЬТУРА ПРОГРАММИРОВАНИЯ

Программисты, составляющие неясные, громоздкие программы, любят говорить в свое оправдание, что это предназначено для компьютера, а не для человека. Конечно, компьютер механически выполняет действия и не интересуется ясностью, наглядностью программы. Однако главным читателем программ является все-таки человек, а не компьютер. При чтении программ человек знакомится с опытом и идеями других программистов, учится сам составлять программы, совершенствует их. Только что написанную программу читать нетрудно, так как многое еще сохраняется и в нашей памяти. Но с течением времени тонкости забываются, поэтому программы должны быть написаны ясно, наглядно и понятно для других.

В учебниках все больше внимания уделяется хорошему стилю программирования: выработано немало правил,



дисплей со световым пером, а также память на магнитных картах. Язык АНАЛИТИК явился расширением АЛМИРа. В АНАЛИТИКе применено модульное построение программ с использованием стандартных фрагментов и процедур. АНАЛИТИК позволяет выполнять аналитические преобразования в буквенном виде: алгебраические действия, взятие производственных и интегралов. Интересно на АНАЛИТИКе организовывались вычисления с переменной разрядностью. АНАЛИТИК реализован также в ЭВМ МИР-3 и СМ-1410. Дальнейшее развитие языка пойдет по пути введения операций обработки текстов.

10-летний опыт работы на МИРах показал высокую эффективность средств общения с этими ЭВМ. Обучение языку программирования и методике работы на ЭВМ требует всего 6—10 часов. Простейшие задачи студенты начинают самостоятельно программировать и решать в течение первого часа знакомства с языком и пультом управления.

Языки АЛМИР—АНАЛИТИК структурного типа и близки к естественному языку, что делает их удобными для описания алгоритмов без графических схем. Программы на этих языках не только хорошо читаются, но и способствуют оперативному творческому мышлению. Для публикаций в нашей литературе можно записать программы только русскими буквами. На практике показано, что составление программы на АЛМИРе и затем ее перевод на ФОРТРАН или ПАСКАЛЬ требует гораздо меньше времени, чем ее составление на этих языках сразу. Например, программа быстрого двумерного преобразования Фурье была составлена на АНАЛИТИКе и отлажена за 3 дня и за 1 день переведена на ФОРТРАН. Такой же программист эту программу на ФОРТРАНе отрабатывал несколько месяцев. Определенную роль здесь играет то, что языки АЛМИР—АНАЛИТИК являются диалоговыми.

Отсутствие в АЛМИРе ограничений на порядок чисел позволяет находить решения с очень большими или очень малыми значениями. На ЕС или СМ ЭВМ такие результаты приводят к переполнению или к исчезновению порядка (в случаях, когда он выходит за пределы ± 75). На МИРе решались задачи с ответами порядка 10^{-1200}.

Опыт показывает, что овладевшие алгоритмизацией и программированием на языках АЛМИР—АНАЛИТИК впоследствии переходят на ФОРТРАН, ПАСКАЛЬ и другие самостоятельно без какой-либо подготовки, используя только справочные сведения по языку.

Литература

1. Глушков В. М. и др. Программное обеспечение ЭВМ МИР-1 и МИР-2. — В 3 т. — Киев: Наукова думка, 1976.

2. Пономарев В. А. Программирование для ЭЦВМ-1. — М.: Сов. радио, 1975.

3. Осипов Л. А. Язык АНАЛИТИК и его сравнение с языками АЛГОЛ и ФОРТРАН. — М.: Наука, 1982.

помогающих писать наглядные программы.

На предыдущих уроках говорилось о подборе смысловых имен. Это один из многих элементов культуры программирования, признак хорошего стиля. Подбор смысла имен улучшает наглядность программы. Однако слишком длинные имена загромождают текст программы. Если сократим их, то смысл становится неясным. В таких случаях удобно вставить дополнительный текст, который не оказывал бы влияния на выполнение программы, но облегчал ее чтение. Такой текст называется комментариями. Комментариями можно пояснить не только имена переменных, но и отдельные части программы, указывать, что выполняет то или иное предложение, и т. п. Комментарии можно вставить повсюду между отдельными символами, словами, числами, именами. Они записываются между скобками (* и *), компьютер на них не реагирует.

Комментарии помогают быстро и легко читать программы. Однако ими не следует злоупотреблять — комментарии должны быть лаконичными, строгими, кратко определять существенное, не загромождать текст.

Поговорим еще об одном элементе культуры программирования — о редактировании программ. Редактированием называется изложение текста программы на бумаге. Гораздо удобнее читать наглядно изложенную программу. В такой программе бывает меньше ошибок, легче найти их и исправить.

Успех редактирования зависит от самого программиста. На всех уроках мы использовали только отредактированные программы, придерживались некоторых общих правил. Например, предложения, находящиеся в другом предложении, отодвигали вправо на несколько позиций, слова **begin** и **end** писали одно под другим, условное предложение изложили так, чтобы удобно выделялись слова **if, then, else.**

П р и м е р. Составим программу для популярной задачи, которую в 1202 г. сформулировал итальянский математик Фибоначчи.

Пара кроликов каждый месяц приносит двух крольчат (самца и самку), а те через два месяца дают новый приплод. Сколько кроликов будет через год, если сначала была одна пара взрослых кроликов?

Из формулировки задачи видно, что к концу первого месяца будет две пары кроликов. К концу второго месяца приплод даст лишь первая пара, поэтому мы получим три пары, но еще через месяц приплод дадут и первая пара, и та, которая появилась два месяца назад. Итого — пять пар.

Символом $F(n)$ обозначим число пар, полученных через n месяцев. Очевидно, что к концу месяца n будем иметь столько пар, сколько их было в предыдущем месяце, т. е. $F(n-1)$, и еще столько новых пар, сколько их было два месяца назад, т. е. в конце месяца $(n-2)$. Получается следующая зависимость: $F(n)=F(n-1)+F(n-2)$.

Приводим программу для определения числа кроличьих пар, спустя n месяцев.

```
program fibonacci;
   var fn, (*F(n)*)
       fn1, (*F(n-1)*)
       fn2, (*F(n-2)*)
       n, (* число месяцев *)
       k: integer;
begin
   fn 1:=1;
   fn:=1; (*F(o)*)
  read (n);
  for k:=1 to n do
      begin
       fn2:=fn1:(*прошел*)
       fn1:=fn;(*один*)
       fn:=fn1+fn2(*месяц*)
       end;
   write (fn)
end.
```

Когда исходное данное равно 12, компьютер напечатает 377. Значит, спустя 12 месяцев получим 377 пар кроликов.

Задачи для повторения

26. Напишите программу для нахождения и напечатания всех номеров счастливых билетов в интервале [m; n], (m, n — исходные данные).

27. Исходные данные — последовательность натуральных чисел. Знак конца последовательности — нуль. Напишите программу для нахождения и

напечатания наименьшего и наибольшего члена последовательности. Нуль не считается членом последовательности.

28. Исходные данные — два натуральных числа *a* и *b*, выражающие длины сторон прямоугольника. Составьте программу для определения всех квадратов, стороны которых выражаются натуральными числами, на которые можно разбить данный прямоугольник, если каждый раз отделяется квадрат наибольшей площади. Например, при исходных данных 5 и 2 результат должен быть таким:



29. Напишите программу для нахождения и напечатания всех натуральных чисел, которые равны сумме кубов своих цифр.

Ответы и решения задач

1. a) 8; b) 2; c) 0.
2. c) 40; 41; 42; 43; 44; d) 7.
3. $a=5$, $b=5$.
4. a) $x=7$, $y=7$; b) $x=10$, $y=100$.
5. a)
```
x2:=x*x;
x4:=x2*x2;
x7:=x4*x2*x
```

6. Возможны разные решения. Приведем два:

```
1) b:=a div 100*10+a mod 10
2) x:=a div 100;
   y:=a mod 10;
   b:=x*10+y
```

7. a) 50; b) 10
8. $a=2$; $b=2$
9. a)
```
if x<0
  then y:=-x*x
  else if x mod 2=0
         then y:=x
         else y:=x+1
```

10.
```
nr:=0;
if m>=1896
  then nr if (m-1896) mod 4=0
   then nr:=(m — 1896) div 4+1
```

11.
```
if min 2<min1
  then begin
         min2:=min2+60;
         h2:=h2-1
       end;
min:=min2 — min1;
if h2<h1
  then h2:=h2+24;
h:=h2-h1
```

12.
```
a:=n; сколько:=0;
while a>0 do
  begin
    сколько:=сколько+1;
    a:=a div 10
end
```

13.
```
p:=n; a:=0;
while p>0 do
  begin
    a:=a*10+p mod 10;
    p:=p div 10
  end
```

14.
```
sum:=0;
x:=1;
for i:=1 to n do
  begin
    sum:=sum+x;
    x:=x*10+1
  end
```

15.
```
s:=0;
for i:=1 to n do
  if i mod 2=1
    then s:=s+i*(i+2)
```

16. a) действия внешнего цикла выполняются один раз, внутреннего — 11 раз.

17. Приведем два варианта решения:

```
1) k:=0;
   for a:=1 to 9 do
     for b:=0 to 9 do
       for c:=0 to 9 do
         if b=a+c
           then k:=k+1
2) k:=0;
   for a:=1 to 9 do
     for c:=0 to 8 do
       if a+c<10
         then k:=k+1
```

18. $a=-5$

19. a)
```
ABBB    ABBBBBB
AABB    AABBBBB
AAAB    AAABBBB
AAAA    AAAABBB
        AAAAABB
        AAAAAAB
        AAAAAAA
```

20.
```
U = = = U
U = = = U
U = = = U
U U U U U
```

21. Пропущены условия: *d*<=*x* и *x***x* **mod** *d*=*x*.

22.
```
program площадь;
      var a, b, q;:integer;
begin
   read (q);
   a:=1; b:=q;
   while a<=b do
      begin
         if a*b=q
            then writeln (a, b);
         a:=a+1;
         b:=q div a
      end
end.
```

23. Одно из простейших решений такое:
```
program делимость;
   var n, s, d: integer;
begin
   read (n);
   for s:=1 to n do
      begin
         write (s); write ('  ');
         for d:=1 to s do
            if s mod d=0
               then write ('+');
      writeln
      end
end.
```
Это решение нельзя назвать эффективным, но из формулировки задачи ясно, что *n* не может быть слишком большим, поэтому количество вычислений не будет велико. Так что в этом аспекте нет надобности совершенствовать программу.

24. 50331.

Данное число печатается в обратном порядке (начиная с последней цифры).

25.
```
*
* *
* * *
* * * * *
* * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * *
```

В этой программе звездочками печатаются первые *n*+1 члены последовательности Фибоначчи.

26.
```
program счастливые:
      var m, n, (*интервал*)
         nr: integer;
begin
   read (m, n);
   for nr:=m to n do
      if nr div 100000+
         nr div 10000 mod 10+
         nr div 1000 mod 10=
         nr mod 1000 div 100+
         nr mod 100 div 10+
         nr mod 10
      then write (nr)
end.
```

27.
```
program минмах;
      var мин, мах, n: integer;
begin
   read (n);
   мин:=n; мах:=n;
   while n< > 0 do
      begin
         if n<мин
            then мин:=n
            else if n>мах
                  then мах:=n;
         read (n)
      end;
   write (мин, мах)
end.
```

28.
```
program квадраты;
      var  a, b, (*длины сторон*)
         n: (*число квадратов*)
         x; integer;
begin
   read (a, b);
   n:=0;
   while b< >0 do
         begin
            n:=n+a div b;
            x:=b;
            b:=a mod b;
            a:=x
         end;
   write (n)
end.
```

29. Все натуральные числа компьютер проверить не может. Поэтому сначала нужно установить интервал, в котором можно надеяться найти разыскиваемые числа. Возможен ли такой случай, когда этот интервал конечен?

Самое большое число — 9. Поэтому наибольшая сумма кубов цифр однозначных чисел — 9^3=729, двухзначных — 9^3+9^3=1458, трехзначных — 9^3+ $+9^3+9^3$=2281, четырехзначных — 9^3+

$+9^3+9^3+9^3=2916$. Наибольшая сумма кубов цифр пятизначных чисел — 3645, т. е. только четырехзначное число. Следовательно, пятизначные и последующие числа не удовлетворяют указанному условию, и их не надо проверять. Также можно не проверять и четырехзначные числа, большие 2916. Можно еще более сужать интервал. Заметим, что число, сумма кубов цифр которого наибольшая и которое меньше чем 2916, но больше, чем 2000, является 2899. Сумма кубов цифр этого числа равна $2^3+8^3+9^3+9^3=1978$, т. е. меньше, чем 2000. Следовательно, достаточно проверить числа, большие 2000, нас не интересуют. Достаточно проверить числа в интервале [1;2000].

Теперь приведем составленную программу:

```
program кубы (output);
  var x,  (*исследуемое чис-
            ло*)
  n,  (*последняя цифра*)
  p,  (*число без послед-
            ней цифры*)
  s: integer; (*сумма кубов
            цифр*)
begin
   for x:=1 to 2000 do
      begin
         s:=0;
         p:=x;
         while p>0 do
           (*нахождение суммы*)
           (*кубов цифр числа*)
           begin
              n:=p mod 10;
              p:=p div 10;
              s:=s+n*n*n
           end.
         if x=s
           then writeln (n)
      end
end.
```

Для любопытства приведем и результаты, полученные после выполнения программы:

```
  1
153
370
371
407
```

Обратите внимание на то, что эта программа отличается от других. Она однократная, т. е. не зависит от исходных данных, так что достаточно выполнить ее один раз и получить результаты. Эффективность одноразовых программ не очень важна, следовательно, можно более не сужать исследуемый интервал.

Заключение

Материала этих уроков достаточно для самостоятельного составления программ несложных задач. Если понадобится составить более сложные программы, нужно научиться владеть данными разных типов: логическими, литерными, вещественными (дробными), а также данными, значения которых являются составными — массивами, записями, множествами.

Составить программу сложной задачи сразу трудновато. Поэтому такую задачу разбивают на мелкие части и программу пишут отдельно. Для оформления отдельных частей программ употребляются специальные конструкции программирования — функции и процедуры. Их придется выучить тем, кто дальше будет работать над программированием задач на языке ПАСКАЛЬ.

Когда изучите упомянутые типы данных, функции и процедуры, вы уже будете ознакомлены с основными конструкциями программирования. Желающим ознакомиться с упомянутыми конструкциями программирования, научиться методам разработки программ можно рекомендовать множество книг, выпущенных на русском языке. Некоторые из них приведены в списке литературы.

В книге Г. Григаса [4] изучение структур управления ведется по методике, опробованной в работе Литовской заочной школы молодых программистов. С этой методикой вы только что познакомились, прочитав этот выпуск.

В книге [3] изложены другие, неизвестные пока (здесь не описаны) структуры данных. Ознакомиться со всеми конструкциями языка ПАСКАЛЬ можно в книгах [1, 2, 5, 7]. Две книги М. Вирта [1, 2] представляют собой хороший учебник программирования, обучающий современным методам разработки программ и проверки их правильности. А

всем, кто желает увидеть, как выглядят программы разных интересных задач, советуем поработать с книгой В. Дагене, Г. Григаса и К. Аугутиса [6].

Список рекомендуемой литературы

1. Вирт Н. Систематическое программирование. Введение. — М.: Мир, 1977.
2. Вирт Н. Алгоритмы + структуры данных = программы. — М.: Мир, 1985.
3. Вьюкова Н. М., Галатенко В. А., Ходулев А. Б. Систематический подход к программированию. — М.: Наука, 1988.
4. Григас Г. Начала программирования. — М.: Просвещение, 1987.
5. Грогоно П. Программирование на языке Паскаль. — М.: Мир, 1982.
6. Дагене В., Григас Г., Аугутис К. Сто задач программирования. — М.: Просвещение (в печати).
7. Перминов О. Н. Программирование на языке Паскаль. — М.: Радио и связь, 1988.

Ваши вопросы и замечания присылайте по адресу: 232600, Литовская ССР, Вильнюс, ул. Академийос, 4. Институт математики и кибернетики, а также в издательство «Знание», в редакцию «Научно-технической литературы».

От редакции

В настоящее время издано достаточно много литературы по основам программирования. Спрос на литературу такого толка очень велик, и поэтому редакция предполагает познакомить подписчиков серии «Вычислительная техника и ее применение» с программированием на языках БЕЙСИК и ФОРТРАН. Однако предварительно мы хотели бы узнать, что думают по этому поводу наши читатели. Вопросы:

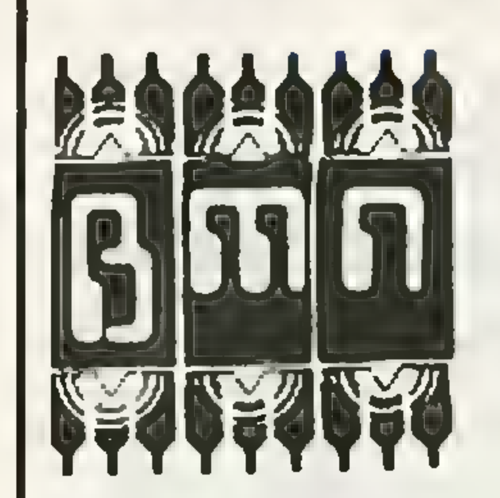
Нужно ли готовить выпуски по основам программирования?

Достаточно ли трех выпусков (ПАСКАЛЬ, БЕЙСИК, ФОРТРАН)?

Какие вопросы, относящиеся к программному обеспечению и программированию, интересуют подписчиков?

Что, по вашему мнению, следует изменить в наших выпусках. Что добавить, от чего отказаться?

Укажите ваш возраст и профессию.



Н. МЕШКОВ
С. УШАНОВ

ЭТИ РАЗНЫЕ ПАСКАЛИ

Подобно естественным языкам, на которых люди общаются друг с другом, высокоуровневые языки программирования имеют обычно множество «диалектов» — версий. Сказанное относится и к языку ПАСКАЛЬ, различные варианты которого реализованы на всех существующих типах ЭВМ. Многообразие версий возникло из естественного желания программистов наиболее полно и эффективно использовать ресурсы различных компьютеров.

ПАСКАЛЬ относится к классу так называемых компилируемых языков программирования. Это значит, что компьютер, выполняя составленную вами на ПАСКАЛЕ программу, станет «переводить» ее на единственно понятный электронному процессору язык машинных команд не «пофразово», а как бы в целом. Результат — гораздо более высокое качество «перевода», как у переводчика-профессионала по сравнению с учеником, переводящим иностранный текст буквально, слово в слово. Прочитав вашу программу, компьютер создает для своей дальнейшей работы «исполняемый» файл (файл — структурная единица в информационном архиве компьютера, так сказать, папка с делом). Этот файл потом и будет играть роль программы. Его можно запускать в работу сразу, без предварительной подготовки.

Для перевода программы из более удобной человеку формы языка высокого уровня на удобный компьютеру язык используются программы-переводчики, или компиляторы. Один и тот же язык программирования (скажем, наш ПАСКАЛЬ) нередко имеет несколько различных компиляторов, отличающихся и самой процедурой компиляции, и использованием конкретных

особенностей компьютера и его операционной системы, т. е. той довольно сложной совокупности служебных программ, что обеспечивает работу ЭВМ и ее взаимодействие с пользователем. Перечислить все версии ПАСКАЛЯ совершенно немыслимо, да и вряд ли это нужно. Но для примера назовем хотя бы пару.

Hi Soft Pascal. Это кассетный вариант компилятора. Все файлы такой Паскаль-системы поочередно загружаются в память компьютера с магнитофона и хранятся в памяти до окончания работы с такой системой. Имеется удобный экранный редактор. Запускается из режима работы с языком БЕЙСИК, куда можно вновь вернуться по окончании использования ПАСКАЛЯ.

Pascal 80. Компилятор, разработанный известной фирмой «Майкрософт» для операционной системы CP/M-80 (она применяется на многих 8-разрядных микрокомпьютерах). Допускает раздельную компиляцию файлов и использование программных модулей, созданных и предварительно прошедших компилирование в иных подобных системах программирования с высокоуровневыми языками СИ, ФОРТРАН и др.

Наиболее популярное ныне семейство Паскаль-систем называется TURBO Pascal. Его разработала фирма «Борлэнд». Этот компилятор поддерживается многими распространенными операционными системами персональных ЭВМ, такими, как CP/M-80, MS DOS, MSX DOS. Его важнейшая особенность — ускоренная однопроходная процедура компиляции, что обеспечивает всей системе высокое быстродействие. Основные действия выполняются в оперативной памяти компьютера (без обращения к дисковым накопителям). Система эта активно развивается: например, пользователи компьютеров типа IBM PC сегодня могут работать уже с пятой версией компилятора TURBO Pascal.

Поскольку в наборе средств стандартного ПАСКАЛЯ нет весьма нужных современному пользователю графических операторов, разработчикам программного обеспечения персональных компьютеров пришлось специально создать пакеты стандартной графики. Хорошим примером может служить графический пакет четвертой версии системы TURBO Pascal, дающий возможность работать с различными графическими адаптерами компьютера IBM PC (Hercules, CGA, EGA и др.).

Система TURBO Pascal для персональных компьютеров стандарта MSX (их дисковая операционная система называется MSX DOS) тоже весьма популярна. Укажем некоторые ее особенности. В ней нельзя, например, независимо компилировать, а затем компоновать вместе отдельные программные модули, тем более если они написаны на различных языках программирования. Допустимы лишь вставки, написанные на низкоуровневом языке АССЕМБЛЕРа.

Имеются, кроме того, незначительные отклонения от стандарта языка ПАСКАЛЬ. Так, стандартные процедуры GET и PUT заменены здесь процедурами READ и WRITE, а процедура PAGE не используется вовсе. Не различаются упакованные и неупакованные типы данных. Есть и некоторые другие отличия.

Но прежде чем приступить к практической работе с языком программирования ПАСКАЛЬ на компьютере стандарта MSX (скажем, «Ямаха-MSX»), снабженном дисковой операционной системой, вам нужно поближе познакомиться с самой этой системой, которую для краткости мы будем называть ниже ДОС.

ДОС — это целый программный комплекс, берущий на себя роль посредника между компьютером и человеком, посредника, обеспечивающего пользователю удобный доступ к аппаратным ресурсам (системному блоку, клавиатуре, монитору, дисководу, принтеру и т. д.), а кроме того, обслуживающий работу других специальных программ (таких, как системы программирования, редакторы текста, базы данных, электронные таблицы и т. п.). Рассматриваемый нами в прошлом выпуске компьютер «Ямаха» работает с операционной системой MSX DOS, созданной фирмой «Майкрософт» по образцу системы CP/M-80. Две эти

системы совместимы на уровне обращения к функциям.

Итак, если вы хотите воспользоваться услугами операционной системы MSX DOS, прежде всего вам нужно будет записать на рабочую дискету два файла: COMMAND. COM и MSX DOS. SYS. Первый файл содержит программы, обслуживающие общение пользователя с системой через набор так называемых системных команд, а второй — подпрограммы управления ресурсами вашего персонального компьютера.

Затем перепишите на дискету саму систему программирования TURBO Pascal. Она состоит из файла компилятора (turbo. com) и файла сообщений об ошибках (turbo. msg). Этот набор файлов — абсолютно необходимый минимум, хотя существуют и иные файлы, расширяющие возможности Паскаль-системы.

Ну теперь вы готовы к работе. Можно начинать. Возьмите подготовленную вами дискету с ДОС, включите компьютер и, не теряя времени, вставьте дискету в щель дисковода. Компьютер прочтет информацию с дискеты, и на экране монитора появится примерно такая заставка:

```
MSX—DOS version 1.03
Copyright 1984 by Microsoft
COMMAND version 1.11
A >
```

Значок «A>» в начале последней строки свидетельствует о готовности компьютера принимать и выполнять системные команды. Напечатайте на экране сразу за этим значком команду: mode 80 — и нажмите клавишу ВК («возврат каретки», она же «ввод»). Эта команда заставляет компьютер перейти в режим работы с 80 символами в каждой строке экрана.

ВНИМАНИЕ: Паскаль-система здесь может работать только в режиме 80-символьной строки. Если вам нравится режим с более крупными буквами (40 символов в строке), этим привычным удобством поневоле придется пожертвовать.

Затем после значка «A>» наберите команду turbo и снова нажмите клавишу ВК. Теперь компьютер отреагирует следующим сообщением на экране:

```
-----------------------------------------------------------
TURBO Pascal system       Version 3.00A
                          CP/M-80, Z80
Copyright (C) 1983,84,85 BORLAND Inc
-----------------------------------------------------------
Terminal: MSX — 2
Include error messages (Y/N)?
```

Последняя строка — адресованный вам вопрос: нужны ли сообщения об ошибках? Если вы ответите буквой Y (Yes, т. е. «Да»), тогда при обнаружении ошибок в программе компьютер станет сообщать вам номер строки, где допущена ошибка, кодовый номер типа ошибки, по которому можно прочесть соответствующее описание в документации, и словесное наименование типа ошибки (например: Syntax error — «Ошибка синтаксиса»). В случае же ответа буквой N (No, т. е. «Нет») вам будет сообщен лишь номер ошибочной строки и номер типа ошибки. Конечно, сразу получить на экране ясную словесную информацию удобнее, однако, отказавшись от такого удобства, вы получаете взамен другое — экономию примерно 2 кбайт оперативной памяти компьютера. Если ваша программа велика, этот дополнительный объем доступной памяти окажется весьма кстати. Выбирайте, что вам важнее.

Теперь на экране монитора появляется такая картинка:

```
Logged drive: A
Work file:
Main file:
 Edit     Compile  Run   Save
 execute  Dir      Quit  compiler  Options
 Text:         0   bytes (83C6—83C6)
 Free:     17471   bytes (83C7—C806)
```

Это «меню», т. е. перечень предоставляемых Паскаль-системой возможностей. В рамках этой системы вы можете: написать и отредактировать программу (команда Edit), произвести ее компилирование (команда Compile), запустить компилированную программу (команда Run), записать ее на дискету для длительного хранения (команда Save), просмотреть каталог содержащихся на дискете файлов (команда Dir) и в конце концов просто выйти из Паскаль-системы, если она вам больше не нужна (команда Quit). Существует целый ряд и других функций системы,

однако на первых этапах они нам вряд ли понадобятся. Поэтому знакомство с ними мы пока отложим.

Любая из перечисленных команд начнет выполняться, если после появления на экране значка готовности «>» (он выполняет здесь ту же роль, что и знак «A>» в дисковой операционной системе MSX) вы нажмете клавишу с первой буквой наименования команды (латинские буквы E, C, R и т. д.).

Собственно процедура создания программы в Паскаль-системе начинается с выбора имени рабочего файла. С этим файлом вы и будете работать — писать его текст, редактировать, а если потребуется, сможете сохранить его на дискете. Давайте проделаем такую начальную процедуру: отвечая на приглашение системы «>», нажмите клавишу с буквой W. На экране монитора немедленно появится надпись Work file name («Имя рабочего файла»). Напечатайте на экране любое слово, каким вы хотели бы назвать свой будущий рабочий файл, например MASTER, — и не забудьте ВК. Вообще говоря, правильное имя файла должно состоять из двух частей — первой (имени) и отделенной от нее точкой второй (что-то вроде фамилии у людей, показывающей некий родовой признак данного файла). Поскольку мы с вами еще не знаем, какая нужна нам вторая часть — она называется расширением, — можно ее просто не писать. В этом случае управляемый Паскаль-системой компьютер сам присвоит названию нашего файла расширение PAS (сокращение от слова Pascal — название языка программирования), и полное имя рабочего файла будет таким: MASTER.PAS.

Выполняя посланную вами команду W, компьютер прежде всего проверит, нет ли в его памяти (оперативной и долговременной — на вставленной в дисковод дискете) файла с тем же именем. Если нет, он напишет на экране NEW FILES («Новый файл»), а если есть, прочтет с дискеты содержимое, перенеся, таким образом, файл в свою быструю оперативную память. Чтобы начать редактирование найденного или вновь заводимого файла, требуется нажать клавишу с буквой E (латинской). При переходе в режим редактирования экран монитора очищается от прежних записей и в верхней строке появляются слова — указатели текущего состояния:

Line — за этим словом будет следовать номер экранной строки, где в данный момент находится курсор (строки считаются сверху вниз, начиная с нуля);

Col — за этим словом будет следовать номер колонки, где в данный момент находится курсор (колонки считаются слева направо, начиная с нуля);

Insert — признак включения режима вставки знаков или группы знаков в текст программы (при нажатии клавиши Ins компьютер переключается в режим замены фрагментов текста, и вместо слова Insert появляется слово Overwrite, а повторное нажатие той же клавиши возвращает прежний режим);

Indent MASTER.PAS — имя редактируемого в данный момент файла.

Работая в режиме редактирования, вы можете составлять свою программу. После того как первая строка программы на экране будет завершена, нажмите клавишу ВК — курсор при этом перейдет на начало следующей строки. Можно теперь записать вторую строку программы и т. д.

Назовем еще две распространенные команды, использование которых облегчает подготовку текста программ:

CTRL+L — удаление ненужной строки,

CTRL+O — освобождение между строками программы места для записи новой строки.

Примененное здесь нами обозначение типа CTRL+L подразумевает вот что: клавиши CTRL и L должны быть нажаты одновременно (служебная клавиша CTRL меняет значение алфавитно-цифровых клавиш). Разумеется, никакой спринтерской сноровки в идеально одновременном ударе по двум клавишам не требуется — достаточно сперва нажать CTRL и, не отпуская ее, нажать клавишу с буквой. Но не наоборот: если вы вначале ударите по буквенной клавише, компьютер, не знающий ваших намерений, попросту напечатает на экране соответствующую букву.

Удалить из текста программы оши-

бочно напечатанный символ или целую группу идущих подряд ошибочных символов можно при помощи служебной клавиши DEL.

Закончив составление и редактирование программы, дважды нажмите клавишу ESC. Тем самым вы выведете компьютер из режима редактирования программы вновь в общий режим Паскаль-системы. Подтверждение тому — вновь возникший на экране значок «>». Нажмите теперь клавишу с буквой С (латинское). Этим вы послали уже упоминавшуюся команду из основного меню (Compile), заставляющую компьютер приступить к процедуре компилирования только что составленной вами программы. Компиляция будет продолжаться до ее завершения или до первой обнаруженной компьютером ошибки в программе. Обнаружив такой дефект, машина приостанавливает компилирование и сообщает вам номер строки, содержащей ошибку, и номер типа ошибки (или даже краткое словесное его описание — если ранее вы не отказались от файла сообщений об ошибках ради экономии памяти). Компилирование продолжаться не может, и вам придется нажать клавишу ESC, чтобы вернуться в режим редактирования текста программы, найти и исправить обнаруженную компьютером погрешность. Тут машина, а точнее, ПАСКАЛЬ система, еще раз приходит вам на помощь: то место в тексте, где обнаружена ошибка, будет отмечено курсором. Когда вы закончите внесение исправлений, снова дважды нажмите клавишу ESC для продолжения процедуры компилирования.

И вот наконец компилирование успешно завершено. Это будет отмечено появлением на экране привычного уже вам значка «>». Теперь ничто не препятствует запуску программы на исполнение. Нажмите клавишу с буквой R (команда Run) — и все.

Если, скажем, вы составляли программу вычисления факториала (этот пример рассмотрен нами ниже), то компьютер напишет на экране следующую заставку:

Программа вычисления функции F(n)=n!
n=

Наберите на клавиатуре какое-нибудь число — положим, 5. Нажмите клавишу ВК, и вы тут же увидите ответ:
F(5) = 120
Под ответом на экране вновь возникнет знак «>». То есть компьютер исполнил программу до конца и опять ждет ваших дальнейших указаний. Конечно, если бы программа вычисления факториала была составлена иначе — не заканчивалась бы после выполнения однократного расчета, а возвращалась бы к своему началу и просила бы пользователя ввести следующее число, исполнение программы на этом бы не завершилось и компьютер бы оставался «внутри» программы, не выходя за «охватывающую» ее Паскаль-систему.

Когда вы опробуете составленную и откомпилированную программу в действии и убедитесь, что она работает как следует, вам необходимо будет записать ее на дискету для хранения, поскольку из оперативной памяти машины она исчезнет при выключении питания. Нажмите клавишу с буквой S (команда Save) и подождите короткое время, пока не закончится запись, — дисковод перестанет урчать и лампочка на его передней панели погаснет. Таковы главные процедуры работы с Паскаль-системой. Если она вам уже надоела и хочется перейти в ее «оболочку» — дисковую операционную систему MSX DOS, — нажмите клавишу с буквой Q (команда Quit). Приятная особенность: если вы по неопытности забыли записать свою программу на дискету и торопитесь покинуть Паскаль-систему, компьютер как всегда любезно напомнит вам об оплошности и попросит подтвердить, что вы действительно не желаете на этот раз сохранять плод своих трудов.

В заключение этого краткого описания свойств системы TURBO Pascal, работающей под управлением MSX DOS, хочется сделать еще одно-два замечания общего характера. Нередко случается так, что составленная вами программа запускается и вроде бы работает, однако дает неверные результаты расчетов. Рекомендуем в такой ситуации тщательно проверить текст программы, поскольку в ней скорее всего оказались логические ошиб-

ки. Компьютер не в состоянии обнаружить погрешности подобного типа: ход ваших мыслей ему неизвестен, и, если формальных нарушений правил составления программ (вроде синтаксических ошибок) нет, машина ничем помочь вам тут не сможет. Внимательно проверьте все логические построения вашего алгоритма, а для исправления найденных погрешностей вновь воспользуйтесь режимом редактирования и потом снова запустите исправленную программу для проверки результатов ее работы.

Добавим еще вот что: команда Run (клавиша с буквой R), запускающая выполнение программы на компьютере, в качестве начальной стадии включает в себя процедуру компилирования. Поэтому отдельно команду Compile можно и не применять.

Из чего состоят программы

До сих пор мы говорили с вами главным образом о технике обращения с основными средствами Паскаль-системы, т. е. знакомили читателей с элементарными правилами написания, запуска, сохранения программ. Не зная этих правил, работать на компьютере невозможно. Однако усвоение лишь этих чисто технических приемов позволит составлять простейшие программы, но не даст вам представления о методах хорошего программирования. А это качество для программиста чрезвычайно важно. Составляя достаточно сложную программу, вы непременно запутаетесь в клубке многочисленных логических ходов, даже пока вся схема программы у вас еще свежа в голове. А что будет, когда вы захотите вернуться к программе через какое-то время? Разобраться даже в собственных записях — особенно если они составлены бессистемно, без четкой структуры, просто неаккуратно — по истечении всего нескольких месяцев будет трудно. О том свидетельствует опыт. Вот почему для облегчения составления четких и безошибочных программ «отцы программирования» рекомендуют усвоить и всегда применять на практике определенные структурные принципы. Скажем об этих принципах несколько слов и ниже проиллюстрируем их использование рядом примеров, предложив читателям несколько простых программ, написанных на языке ПАСКАЛЬ. Кроме собственно текста программы, для облегчения понимания мы приводим и четкий логический «костяк» — то, что называется алгоритмом. В практическом программировании не все прибегают к предварительной записи алгоритма, однако никому еще не удавалось обойтись без представления такого алгоритма вообще, хотя бы в уме, если не на бумаге. Так что не экономьте бумагу (хоть у нас ее и мало) столь неудачным способом: пожалев листок на схему алгоритма программы, вы непременно потеряете десятки, а то и сотни листов и часы работы в ходе поиска коварных ошибок.

Написание надежных, удобных, легко читаемых и понимаемых программ обычно связывают с соблюдением главных принципов структурного программирования. Принципы эти основаны на использовании следующих четырех базовых управляющих структур:

— последовательные действия,
— альтернативные действия,
— повторяемые действия,
— группирование.

Основной управляющей структурой любого языка программирования является последовательность инструкций, т. е. элементарных приказов компьютеру. Порядок выполнения инструкции здесь будет задаваться ее местом в тексте программы, так как ЭВМ исполняет инструкции последовательно — строго одну за другой.

Любой отрезок общей последовательности инструкций, выделенный внутренним логическим смыслом совокупности его процедур, в языке ПАСКАЛЬ принято заключать между особыми служебными словами begin и end («начало» и «конец»). Их иногда называют операторными скобками (begin — «открывающая скобка», end — «закрывающая»), потому что они выделяют часть последовательности инструкций, как обычные скобки часть фразы или математического выражения. Вот общий вид подобной конструкции:

begin

⟨инструкция 1⟩;
. . . .
⟨инструкция n⟩;
end;

Эта управляющая структура именуется группированием. Выделенную таким способом группу инструкций компьютер рассматривает как единое целое и исполняет ее как единую инструкцию. Возможность группирования инструкций значительно повышает эффективность использования других управляющих структур.

Управляющая структура, названная альтернативными действиями, дает возможность выбора: выполняться будут либо те, либо иные действия в зависимости от соблюдения некоторых условий (точнее, от значения соответствующей логической переменной). В языке ПАСКАЛЬ альтернативные действия организуются с помощью инструкции if («если»):

Алгоритм	**Фрагмент программы**
ЕСЛИ ⟨условие⟩ ТО	if ⟨условие⟩ then
⟨действие 1⟩	⟨инструкция 1⟩
ИНАЧЕ	else
⟨действие 2⟩	⟨инструкция 2⟩;
ВСЕ	

Здесь мы привели обещанную выше двойную запись: слева — алгоритм (костяк логический), справа — подлинный фрагмент программы на языке ПАСКАЛЬ с подобающим синтаксисом. В алгоритмической записи английские служебные слова заменены их подходящими русскими эквивалентами. Угловыми скобками обозначены «непостоянные», подвижные части, вместо которых в каждом конкретном случае нужно подставить описания реальных действий или реальные инструкции для компьютера.

Записанная выше конструкция именуется простой альтернативой. При помощи упоминавшихся уже операторных скобок можно построить и сложную альтернативу:

Алгоритм	**Фрагмент программы**
ЕСЛИ ⟨условие⟩ ТО	if ⟨условие⟩ then
	begin
⟨действия 1⟩	⟨инструкции 1⟩
ИНАЧЕ	end
	else
⟨действия 2⟩	begin
ВСЕ	⟨инструкции 2⟩
	end;

Как вы видите, в таком случае альтернативные действия состоят не из какой-то одной инструкции, а из целой их группы, заключенной между операторами.

Еще более сложную конструкцию можно соорудить, вкладывая альтернативные действия друг в друга — наподобие слоеного пирога:

Алгоритм	**Фрагмент программы**
ЕСЛИ ⟨условие 1⟩ ТО	if ⟨условие 1⟩ then
	begin
⟨действия 1⟩	⟨инструкции 1⟩
ИНАЧЕ	end
	else
ЕСЛИ ⟨условие 2⟩ ТО	if ⟨условие 2⟩ then
	begin
⟨действия 2⟩	⟨инструкции 2⟩
	end
ИНАЧЕ	else
⟨действия 3⟩	begin
ВСЕ	⟨инструкции 3⟩
	end;
ВСЕ	

В качестве условий, определяющих поведение компьютера при выполнении альтернативных действий, используются логические выражения (в частном случае — просто логические переменные), могущие принимать одно из двух значение: ДА (истина) либо НЕТ (ложь).

Для иллюстрации сказанного приведем пример составления простенькой программы на ПАСКАЛЕ.

Программа нахождения корней квадратного уравнения. Дано квадратное уравнение в общем виде: $AX^2+BX+C=0$, даны числовые значения его коэффициентов А, В и С. Требуется определить его вещественные корни X1 и X2.

Рассмотрим все мыслимые разветвления, обусловленные различными сочетаниями значений коэффициентов:

если $A=B=C=0$, то X1 и X2 — любые вещественные числа (бесконечное множество решений);

если $A=B=0$, но $C\neq0$, тогда решений нет;

если $A=0$, $B\neq0$, то $X=-C/B$ (единственное решение);

если $A\neq0$ и $B^2-4AC=0$, то $X1=X2=-B/2A$ (кратные корни);

если $A\neq0$ и $B^2-4AC>0$, то $X1=(-B-\sqrt{B^2-4AC})/2A$, $X2=(-B+\sqrt{B^2-4AC})/2A$;

если $A\neq0$ и $B^2-4AC<0$, то решения нет (отсутствуют вещественные корни).

Мы перебрали все возможные сочетания значений коэффициентов и каждый случай должны отразить в алгоритме, соответственно «объяснив» компьютеру в программе, как определить тип сочетания и какие действия предпринимать в каждом случае:

АЛГОРИТМ

```
ПЕЧАТЬ (Программа решения квадрат-
        ных уравнений
ПЕЧАТЬ (Коэффициенты: A, B, C?)
ВВОД (A, B, C)
ПЕЧАТЬ (Уравнение:
        <A>*X²+<B>*X+<C>=0)
ЕСЛИ A=0 ТО
  ЕСЛИ B=0 ТО
    ЕСЛИ C=0 ТО
      ПЕЧАТЬ (Бесконечное множест-
              во решений)
    ИНАЧЕ
      ПЕЧАТЬ (Решения нет)
    ВСЕ
  ИНАЧЕ
    X=-C/B
    ПЕЧАТЬ (Единственное решение)
    ПЕЧАТЬ (X=<X>)
  ВСЕ
ИНАЧЕ
  D=B²-4*A*C
  ЕСЛИ D=0 ТО
  X=-B/(2*A)
  ПЕЧАТЬ (Кратные корни)
  ПЕЧАТЬ (X1=X2=<X>)
ИНАЧЕ
    ЕСЛИ D>0 ТО
      X1=(-B-JD)/(2*A)
      X2=(-B+JD)/(2*A)
      ПЕЧАТЬ (Вещественные корни)
      ПЕЧАТЬ (X1=<X1>, X2=<X2>)
    ИНАЧЕ
      ПЕЧАТЬ (Вещественных корней
              нет)
    ВСЕ
  ВСЕ
ВСЕ
КОНЕЦ
```

```
Program prim1;{Программа решения ква-
дратных уравнений}
  var a, b, c, d, x, x1, x2: real;
  begin
    writeln ('Программа решения ',
    'квадратных уравнений');
    write ('Коэффициенты: A, B, C? ');
readln (a, b, c);
writeln ('Уравнение: (',a:5:2,
')*X²+(',b:5:2,
')*X+(',c:5:2,')=0');
if a=0 then
  begin
    if b=0 then
      begin
        if c=0 then
          writeln ('Бесконечное ',
          'множество решений')
        else
          writeln ('Решения ',
          'нет');
      end
    else
      begin
        x:=-c/b;
        writeln ('Единственное ',
        'решение:');
        writeln ('X=',x:6:3);
      end;
  end
    else
      begin
        d:=sqr (b)-4*a*c;
        if d=0 then
          begin
            x:=-b/(2*a);
            writeln ('Кратные ',
            'корни:');
            writeln ('X1=X2=',
            x:6:3);
          end
        else
          if d>0 then
            begin
              x1:=(-b-sqrt (d))
              /(2*a);
              X2:=(-b+sqrt (d))
              /(2*a);
              writeln ('Вещест',
              'венные корни:');
              writeln (' X1=',
              x1:6:3,'          ',
              'X2=',x2:6:3);
            end
          else
            writeln ('Веществен',
            'ных корней нет');
        end;
 end.
```

Многократное повторение однотипных процедур (повторяемые действия) организуют при помощи так называемых операторов цикла. Циклы бывают

такими:

— цикл со счетчиком,
— цикл по условию,
— цикл до выполнения условия,
— цикл с выходом по условию.

Цикл со счетчиком:

АЛГОРИТМ

```
ОТ k=⟨выражение 1⟩ ДО ⟨выражение 2⟩
   ЦИКЛ
      ⟨действия⟩
   ВСЕ
```

ФРАГМЕНТ ПРОГРАММЫ

```
for k:=⟨выражение 1⟩ to ⟨выражение 2⟩ do
begin
   ⟨инструкции⟩
end;
```

Переменная k называется параметром цикла, а группа инструкций, заключенная в операторные скобки begin—end, — это «тело цикла» (т. е. само повторяемое действие). Значения выражений 1 и 2 задают диапазон изменения параметра цикла. Ключевое слово to определяет цикл с возрастающим параметром, а слово downto будет определять цикл с убывающим параметром. На каждом проходе цикла параметр будет изменяться автоматически (программировать это изменение не требуется).

Пример. Программа вычисления факториала:

```
Program fact; { Программа вычисления функции F(n)=n! }
  var n, k, f: integer;
  begin
    writeln ('Программа вычисле',
    'ния функции F(n)=n!');
    write ('n='); readin(n);
    f:=1;
    for k:=1 to n do f:=k*f;
    writeln ('F(',n,')=',f);
  end.
```

Используя в программе циклы такого рода, мы должны точно указывать границы изменения параметра цикла: от такого-то до такого-то значения. Поэтому такие циклы нельзя применять в тех случаях, когда число повторений нам заранее не известно, — оно зависит от результатов, получаемых в самом процессе выполнения цикла. Тогда целесообразно использовать «условные» циклы.

Цикл по условию:

Алгоритм	**Фрагмент программы**
ПОКА ‹условие›	while ‹условие› do
ЦИКЛ	begin
⟨действия⟩	⟨инструкции⟩
ВСЕ	end;

Число повторений такого цикла определяется значением выражения ⟨условие⟩: цикл станет повторяться, пока это выражение сохраняет свое логическое значение «истина». Если с самого начала его значение «ложь», цикл не будет выполнен ни одного раза.

Цикл до выполнения условия:

Алгоритм	**Фрагмент программы**
ЦИКЛ	repeat
⟨действия⟩	⟨инструкции⟩
ДО ⟨условие⟩	until ⟨условие⟩;
ВСЕ	

Число повторений этого цикла также определяется логическим значением выражения ⟨условие⟩. Однако в данном случае цикл будет повторяться до тех пор, пока значением не окажется «истина». Тело такого цикла в любом случае выполняется хотя бы один раз, поскольку проверка условия производится не в начале, а в конце «прохода». В этом цикле тело может состоять из нескольких инструкций — они разделяются точкой с запятой (использовать операторные скобки не требуется).

Цикл с выходом по условию:

Алгоритм

```
ЦИКЛ
   ⟨действия 1⟩
   ЕСЛИ ⟨условие⟩ ТО ВЫХОД
   ИНАЧЕ
      ⟨действия 2⟩
   ВСЕ
ВСЕ
```

Фрагмент программы

```
var out: boolean;
. . . . . . . . . . . . .
  out:=false;
  repeat
    ⟨инструкции 1⟩;
    if ⟨условие⟩ then
    out:=true
      else
        begin
```

```
        ⟨инструкции 2⟩
      end;
  until out;
```

Это наиболее общий тип цикла: его структура позволяет программисту организовать проверку выполнения условия в любом месте тела цикла, а не только в начале или в конце. Цикл будет выполняться до тех пор, пока выражение ⟨условие⟩ не примет логическое значение «истина».

Приведем практический пример использования циклов.

Программа вычисления числа π. Требуется рассчитать значение π с заданной точностью ε.

Воспользуемся следующим представлением π бесконечным рядом:

$$\pi = 4(\underbrace{1 - \frac{1}{3}}_{a1} + \underbrace{\frac{1}{5} - \frac{1}{7}}_{a2} + \ldots) = \sum_{k=1} a_k (k = 1, 5, 9, \ldots),$$

где $a_k = 4\left(\frac{1}{k} - \frac{1}{k+2}\right)$.

Чтобы обеспечить заданную точность, необходимо проверять значение переменной a_k и продолжать повторяющиеся вычисления суммы ряда до тех пор, пока на некотором шаге эта переменная не окажется меньше заданной погрешности ε. Тогда вычисления следует закончить

Алгоритм

```
ПЕЧАТЬ (Программа вычисления
        значения числа π)
ПЕЧАТЬ (Укажите точность вычи-
        слений)
ПЕЧАТЬ (eps=)
ВВОД (eps)
pi=0.0
k=1
ЦИКЛ
  ak=4/k−4/(k+2)
  pi=pi+ak
  ЕСЛИ ak<eps ТО ВЫХОД
  ИНАЧЕ
    k=k+4
  ВСЕ
ВСЕ
ПЕЧАТЬ (Результат: π=<pi>)
КОНЕЦ
```

```
Program PI;{Программа вычисления
значения числа π}
   var k: integer;
      pi, eps, ak: real;
      out: boolean;
   begin
   writeln ('Программа вычисления',
   ' значения числа π');
   writen ('Укажите точность',
   ' вычислений');
   write ('eps='); readln (eps);
   pi:=0.0;
   k:=1;
   out:=false;
   repeat
     ak:=4/k−4/(k+2);
     pi:=pi+ak;
     if ak<eps then
       out:=true
     else
       k:=k+4;
   until out;
   writeln ('Результат: π=',
   pi:10:7);
 end.
```

Дополнительная литература

1. Перминов О. Н. Программирование на языке Паскаль. — М.: Радио и связь, 1988. — 224 с.

2. Брябрин В. М. Программное обеспечение персональных ЭВМ. — М.: Наука, 1988. — 272 с.

3. Программное обеспечение микро-ЭВМ: Практическое пособие для инженерно-педагогических работников системы профессионально-технического образования. Кн. 11. Практикум по программированию. — М.: Высшая школа, 1988. — 143 с.

4. Программное обеспечение микро-ЭВМ: Практическое пособие для инженерно-педагогических работников системы профессионально-технического образования. Кн. 7. Программирование на языке ПАСКАЛЬ. — М.: Высшая школа, 1988. — 126 с.

Персоналии



ДЕЛО ЖИЗНИ — ИНФОРМАТИКА

Декабрь 1988-го стал последним в недолгой жизни Андрея Петровича Ершова — крупного ученого и талантливого организатора.

Андрей Петрович Ершов — один из тех ученых, которые росли вместе с Сибирским отделением АН СССР, чья деятельность создавала авторитет и научную известность работам СО АН СССР. Он принадлежит к первому в советских вузах выпуску специалистов по программированию. А. П. Ершов окончил Московский универстет в 1954 г., затем работал в Вычислительном центре АН СССР заведующим лабораторией автоматизации программирования. В 1958 г. академик С. Л. Соболев, один из создателей СО АН СССР, привлекает молодого ученого к работе по организации научного центра — А. П. Ершов формирует коллектив программистов Института математики СО АН СССР и осуществляет научное руководство работами этого коллектива. Полностью в Сибирское отделение АН СССР Андрей Петрович переходит в 1960 г. К этому времени он был уже хорошо известным ученым — создал один из первых советских трансляторов (тогда они назывались программирующими программами), в котором им был предложен ряд методов и подходов, ставших потом общепринятыми; его перу принадлежит первая советская монография по автоматизации программирования, которая была сразу же после выхода в свет переведена и издана в США, Англии, Китае.

Созданный А. П. Ершовым отдел программирования Института математики СО АН СССР был первым в Сибири коллективом программистов, который внес заметный вклад в развитие вычислительного дела в стране. А. П. Ершов нацелил работу коллектива на решение фундаментальной и весьма важной в то время проблемы — создание оптимизирующего транслятора для языка высокого уровня. Эта новая (и по тогдашнему мнению большинства специалистов, неразрешимая) проблема была решена созданием известной системы программирования АЛЬФА, имевшей большое теоретическое и практическое значение для развития советского программирования. Работа приобрела международную известность, монография по системе АЛЬФА была переведена и издана в США.

Уже в этой работе проявилась одна важная черта А. П. Ершова как ученого — умение осмыслить общую практику программирования, выделить фундаментальные и перспективные направления, найти подходы к решению назревающих проблем.

Научная биография А. П. Ершова неотделима от становления в Советском Союзе системного и теоретического программирования. Его работы способствовали зарождению и развитию этих направлений, формированию их научной и методической платформы.

Конкретный вклад А. П. Ершова в развитие теоретического программирования заключается в получении ряда фундаментальных результатов по теории схем программ, в частности аксиоматике схем Янова и по теории распределения памяти. В исследованиях по системному программированию им была разработана общая методика оптимизирующей трансляции, включающая в себя понятие смешанной стратегии при декомпозиции сложных конструкций алгоритмических языков, выделение машинно-независимого вну-



треннего языка. Широко известны его работы по практической реализации распределения памяти. Под его руководством была создана первая в СССР развитая система разделения времени АИСТ-0, реализованная на многопроцессорной вычислительной системе. Опыт АИСТ-0 оказал важное влияние на последующие отечественные работы по вычислительным центрам коллективного пользования.

Обобщая свой опыт руководства большими программными проектами, Андрей Петрович заложил основы раз-



работки научных методов организации коллективов программистов, а также сформулировал ряд общих принципов программирования как нового и своеобразного вида научной деятельности.

Одна из наиболее интересных его работ «О человеческом факторе в программировании» получила широкую международную известность, неоднократно издавалась за рубежом. Это направление деятельности А. П. Ершова способствовало становлению исследований по технологии программирования в СССР.

Для научной деятельности А. П. Ершова характерно сочетание теоретических исследований и практических работ. Практику системного программирования он умел использовать для постановки новых теоретических задач, а полученные теоретические решения применялись в его работах для создания программных инструментов. Весьма показательна в этом отношении монография «Введение в теоретическое программирование», которая носит подзаголовок «Беседы о методе». Это умение сочетать теорию и практику А. П. Ершов прививал и своим ученикам. В отделах ВЦ СО АН СССР, где до последних дней трудился ученый, одновременно с работами по трансляции велись исследования по теории схем программ и теории оптимизации программ, а теоретические исследования по параллельному программированию служат основой для разработки архитектуры перспективных высокопроизводительных ЭВМ. Благодаря своему умению увидеть новые перспективы в программировании А. П. Ершов всегда выступал как инициатор новых направлений исследований в информатике — таких, как методы использования естественных языков для взаимодействия с ЭВМ, школьная информатика, автоматизация редакционно-издательской деятельности, которые продолжают развиваться в отделе информатики ВЦ СО АН СССР и по сей день.

Отдавая должное научной и организационной работе, особенно важной в период становления новых научных направлений, А. П. Ершов уделял им много внимания. Он был председателем программных комитетов многих международных и всесоюзных конференций. Благодаря его энергии новосибирский Академгородок стал местом проведения ряда важных международных и общесоюзных мероприятий. А. П. Ершов был членом редколлегий ряда ведущих советских и зарубежных журналов, членом научных советов и комиссий по информатике. Он был председателем комиссии по системному математическому обеспечению Координационного комитета по вычислительной технике АН СССР, председателем научного совета по комплексной проблеме «Кибернетика». Большую работу он вел по укреплению международных связей советских программистов. Неоднократно выступал с лекциями в США, Англии, ФРГ, Болгарии, ГДР, Чехословакии. Под его руководством велось научное сотрудничество ВЦ СО АН СССР с рядом зарубежных научных организаций. А. П. Ершов был членом многих рабочих комиссий в Международной федерации по обработке информации (ИФИП).

Академик А. П. Ершов был хорошо известен в стране и за рубежом как создатель и признанный глава Новосибирской школы информатики. Большое значение имела и имеет его деятельность по организации преподавания информатики в школе.

* * *

Судьба распорядилась так, что мне — журналисту по профессии и редактору-консультанту Центрального телевидения по должности, довелось в течение ряда лет находиться с ныне покойным ученым в теснейшем контакте. Именно это обстоятельство дает мне моральное право поделиться с читателями своими воспоминаниями о встречах и работе с Андреем Петровичем.

Воспоминания возвращают меня в 1985 г. Вскоре после принятия известного постановления партии и правительства о введении в школах страны нового предмета «Основы информатики и вычислительной техники» я оказался в кабинете главного редактора. «Необходимо сегодня же приступить к подготовке телевизионного варианта этого предмета. Свяжитесь с академиком Ершовым и приступайте. Только имейте в виду — он живет и работает в новосибирском Академгородке, а это значит — придется раз в квартал (а

может, и чаще) вылетать туда со съемочной аппаратурой».

Предложение необычное. Выступления академиков на телевизионном экране — дело не такое уж редкое, а вот так — академик — ведущий целого сериала — с этим не приходилось еще иметь дела.

Начинаю собирать информацию, телефоны. Наталкиваюсь на записи самого ученого. «...Я имею дело с ЭВМ с 1952 г. и рос среди них, как когда-то сельские дети росли среди лошадей и прочей живности. Прошло время, и инстинктивный рост вместе с развитием ЭВМ сменился поисками профессионального самоопределения. В течение последнего десятилетия эти поиски выразились в виде работ и выступлений на научных собраниях по вопросам взаимодействия человека с ЭВМ, человеческого фактора в программировании, места ЭВМ в развитии человеческого общества».

Сказано более чем скромно. Позже, приступив к работе с Андреем Петровичем, я имел возможность не раз убеждаться в его природной скромности и открытости.

Итак, зима 1985 г., Новосибирск, Академгородок... Съемочная группа размещается в единственной и в своем роде неповторимой гостинице Академгородка — «Золотой долине». Неповторимой в том смысле, что здесь, может быть, как нигде более в Союзе, одновременно собираются крупнейшие ученые со всего света. И вот мы, «телевизионщики», как нас любовно называл Андрей Петрович, в «Золотой долине». Первая очная встреча назначена на следующий день в рабочем кабинете академика.

Дорога от гостиницы к Вычислительному центру Сибирского отделения АН СССР, где в отделе школьной информатики долгие годы работал А. П. Ершов, более чем романтична. Дремучий сосновый бор, извилистая тропинка и ручные белки. Такие дороги запоминаются надолго. Эту дорогу прекрасно знают не только взрослые жители Академгородка, но и дети. Говорят, что самое «жестокое» наказание для ребенка, живущего здесь, — это когда папа в воскресенье не берет его с собой в Вычислительный центр «поиграть» с компьютером. Это своего рода высшая мера наказания.

Кабинет заведующего отделом под стать своему хозяину выглядит на редкость скромно. Никаких изысков, а тем более полированных столов, мягких кресел. Исключительно книжные полки, книги, папки, сувениры, подаренные академику в самых разных странах мира, и единственный портрет. Это портрет его любимого учителя, пионера отечественной кибернетики Алексея Андреевича Ляпунова.

Андрей Петрович исключительно вежлив и предупредителен. «Я все понял. Учиться делать учебные передачи по информатике будем вместе. Завтра к 19.00 буду готов к работе».

Так началась наша совместная работа над телевизионным вариантом курса основ информатики и вычислительной техники для учащихся средней школы.

«Подготовка цикла передач для телевидения оказалась делом, новым для всех. Правда, ощущение неизведанности способствовало быстрому сплочению нашего небольшого творческого коллектива... Дополнительной проблемой оказалась противоречивость требований к организации, постановке и съемке 30-минутных уроков. Трудно сказать, в какой мере эти противоречия удалось примирить друг с другом, но после длительных споров и обсуждений сложился некоторый продуктивный подход... Мы старались избегать «дрессировки» с помощью многократных репетиций, предпочитая менее причесанный, но зато живой и взаимный разговор учеников во время решения задачи. Допущенные ошибки не вырезались из кадра, а подвергались обсуждению с тем, чтобы научить наших зрителей не допускать их в будущем.

Такой импровизированный подход к построению учебных передач потребовал большой и кропотливой работы. Здесь нам очень помогли энтузиазм, живой интерес и изрядная доза терпения дружной команды 9-го «В» класса 166-й школы Советского района Новосибирска, с которой мы отсняли ряд передач. Несмотря на все трудности и

недостаток опыта, работа над учебными программами представляет большой научно-методический интерес.

Информатика еще слишком близка к периоду становления и не настолько «отстоялась», чтобы можно было без больших усилий отфильтровать из ее научного багажа зерна вечных истин, — одновременно глубоких, кратких и в то же время доступных для восприятия молодым умом. Не могу сказать, что мы уже нашли эти формулировки, но работа над серией учебных передач по информатике даст таким поискам мощный стимул». Эти строчки взяты из статьи академика, опубликованной в 1985 г. в газете «Наука в Сибири», т. е. в конце первого полугодия преподавания нового предмета. Поэтому все, включая учебное телевидение, оказались в одинаковом положении: школа впервые приступила к изучению основ информатики, а телевидение впервые училось тому, как учить этому предмету и как показывать его.

К своей работе на телевидении Андрей Петрович относился серьезно, не раз и не два перепроверяя свои записи.

Двумя годами позже, когда этот курс основ информатики с участием академика Ершова уже был завершен, ученый неоднократно возвращался в свое «телевизионное детство». Это был кусок его жизни, не менее важный, чем, скажем, работа над очередным серьезным научным трудом.

Магнитная лента сохранила видеозаписи передачи, в которой после вступительных слов вице-президента АН СССР академика Е. П. Велихова Андрей Петрович с еле уловимой грустью в голосе передает эстафету ведения телепередач машинного курса информатики своему ближайшему сподвижнику А. Г. Кушнеренко. Что поделаешь? У всего в этой жизни есть свое начало и, к сожалению, свой конец...

В моей записной книжке сохранились записи некоторых мыслей ученого, произнесенных им в период нашего творческого содружества.

«Получасовая передача не может вместить систематического изложения всего курса, она не может заменить урок учителя, она не может заменить учебник. Она должна помочь усвоению курса, используя весь арсенал возможностей телевидения».

«Учебная телепередача, проведенная автором учебника, — это ноты, а учитель — исполнитель. И исполнитель должен быть квалифицирован в том смысле, что он должен уметь добросовестно и убедительно провести свою «партию».

«Учитель — это тот же спутник ученого. Разница в одном: ученый добывает истину, а учитель ее воспринимает и несет в умы школьников. Они идут рядом».

Съемки телевизионного варианта основ информатики подходили к концу. Все остальные встречи происходили уже в Москве, и одна из них за пределами страны. Вот о ней-то можно рассказать поподробней.

Жаркий июль 1988 года. Руководство Гостелерадио СССР командирует меня в Венгерскую Народную Республику для участия в Международном конгрессе по обучению математике с использованием средств кино и телевидения.

Аэропорт Будапешта. Объявление по радио на чистом русском языке, и я «в объятиях» приставленного ко мне на время конгресса переводчика. На пути к гостинице останавливаемся недалеко от площади Геллерта, в центре которой — памятник советскому воину-освободителю. Процедура аккредитации занимает всего несколько минут. Тут же узнаю, не ожидается ли участие в работе конгресса академика Ершова из СССР. «Один момент», — отвечает любезная девушка и нажимает клавишу стоящего перед ней персонального компьютера. Принтер печатает мне письменный ответ: гостиница «Фламенко», номер телефона, номер комнаты, дата приезда в Будапешт...

Утром следующего дня набираю номер телефона единственного знакомого мне в ВНР человека. Слышу знакомый голос: «Ершов лисн ю». Я рад безумно, за два года систематических телефонных переговоров «Москва—Новосибирск» Андрей Петрович научился узнавать мой голос с первых интонаций. Но то было дома, а здесь... Набираюсь смелости и в ответ на прекрасный английский говорю на не

менее чисто русском: «Андрей Петрович, будьте проще. Это Илья Семенович...» Однако представляться не следовало, я сразу опознан. «Жду Вас завтра на пленарном заседании. Сегодня работаю допоздна, готовлюсь к докладу, но завтра очень прошу Вас...»

К 9.00 я у входа в зал конгрессов. Люди, собравшиеся здесь, говорят практически на всех языках мира. Не слышу только родного. «Но ничего, — думаю я про себя, — реванш впереди. Сегодняшнего докладчика, пожалуй, один я пойму без посторонней помощи».

Большая толпа организовалась в фойе — участники конгресса добывают наушники для синхронного перевода. Я гордо прохожу мимо и направляюсь в первый ряд.

В президиуме рядом с ведущим или председательствующим из США — академик Андрей Петрович Ершов. Он видит меня и дружески улыбается. Ведущий, открывая пленарное заседание, представляет собравшимся ученого из Советского Союза. Зал взрывается аплодисментами — имя Андрея Петровича знакомо специалистам всего мира. К трибуне подходит академик Ершов и, к моему великому удивлению, начинает: «Леди энд джентльмен...» И дальше — тоже на чисто английском. Тут бы следовало заметить, что мои познания этого международного языка общения ограничиваются программой десятилетки да еще вуза. Ну а что это такое у нас в стране, знают многие.

Только тут я понял всю смехотворность своего положения. Наушники для синхронного перевода надо было всетаки взять...

После окончания заседания Андрей Петрович подходит ко мне и задает вполне естественный вопрос: «Ну, как?» Мне ничего не оставалось — я солгал, высказав вполне определенное свое восхищение докладом. Договорились о встрече на следующий день.

Итак, следующий день, который волею судеб стал последним в нашем трехлетнем творческом содружестве. Мы гуляем по Будапешту. Андрей Петрович весел. Ни единым словом, ни намеком не касается темы здоровья. О тяжелой болезни Андрея Петровича я знал давно.

Город этот Андрей Петрович знает прекрасно. Мы бродим по улицам и площадям, заходим в книжный магазин. Книги — одна из его «слабостей». В магазине я теряю своего попутчика как минимум на час и встречаю его на выходе в состоянии полного счастья. Куплены редкие издания книг и грампластинки. А тут еще «нечаянная радость» — прямо напротив книжного магазина идет бойкая торговля вареной кукурузой. Андрей Петрович по-мальчишески прыгает от радости. Чуть позже я узнаю, что сам вид, но особенно запах сваренного кукурузного початка возвращает его в детство — в прекрасное, беззаботное время ныне уставшего от непомерных нагрузок ученого.

Остаток дня прошел в разговорах о ближайших планах. Андрей Петрович смотрел в будущее уверенно.

«Мечтаю следующим летом отправиться в путешествие по Дунаю, и обязательно с женой. Как-то так получалось, что за всю совместную жизнь нам ни разу не удалось толком отдохнуть вместе: то работа, то бесконечные командировки, потом внуки...»

Перевожу разговор в несколько другое русло. «Коль скоро Вы заговорили о будущем, — говорю я, — давайте попробуем заглянуть в это будущее — каким Вы себе представляете кабинет информатики двухтысячного года и урок в нем?» Ответ был таким: «Ну, я бы свою мечту выразил в несколько парадоксальной форме. Я мечтаю о том, чтобы в двухтысячном году или в какое-то подходящее время кабинет информатики как понятие перестал бы существовать, а ЭВМ была бы такой же естественной принадлежностью каждой парты, как сейчас, скажем, скамейка, розетка, мел, доска... Полное внедрение. И в связи с этим исключение всякой специфики и своеобразия в использовании вычислительной техники».

И снова разговоры о ближайших планах. Одна из общественных организаций, зная о нашем тесном сотрудничестве, обратилась ко мне с предложением подготовить серию видеороликов под условным названием «Компьютер для всех». Причем инициатива исхо-

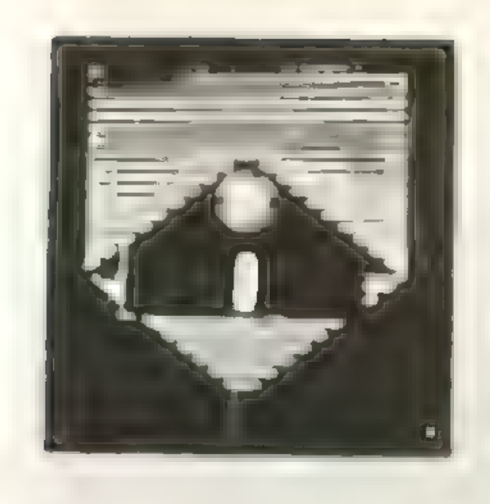
дила от наших зарубежных партнеров, которые уже давно хотели иметь такой сериал с участием академика Ершова. Эти фильмы в популярной форме, с включением видеоклипов, концертных номеров, рекламы вычислительной техники должны стать своего рода руководством к овладению компьютерной грамотой. Имелось в виду, что ведущим таких фильмов должен быть Андрей Петрович и озвучены они будут на русском и английском языках. Академик к предложению отнесся с большим интересом, обещал все обдумать и назвать сроки начала работ. На этом мы и простились. Как выяснилось чуть позже, простились навсегда...

В заключение приведу выдержку из рассказа Андрей Петровича о своем учителе А. А. Ляпунове.

«Роль Алексея Андреевича Ляпунова в науке можно охарактеризовать очень просто. Он один из основателей советской школы информатики и кибернетики. Я горжусь тем, что мне довелось быть докладчиком на заседании первого в Советском Союзе семинара по кибернетике, на котором я рассказал о машинном моделировании условных рефлексов. Надо сказать, что тематика этого доклада очень выразительным способом сфокусировала в себе совокупность разнообразных интересов Алексея Андреевича. Он был создателем советской школы теоретического программирования, через которую прошли, по существу, все ученые, заложившие основы этой науки. Влияние Ляпунова на науку и ученых было связано с его широкими интересами и глубокими познаниями во многих науках, оно объяснялось также и его личными качествами. При этом я имею в виду не только его гражданскую позицию как ученого, очень горячего, принципиального патриота и просто мужественного человека, но и его очень высокую демократичность, интеллигентность, не говоря уже о личном обаянии. Мне припоминается, с каким вниманием Алексей Андреевич относился к выбору правильного научного направления. Помню, однажды я написал техническую статью, и она у меня, как это сказать, ловко получилась. Алексей Андреевич ощутил это мое упоение успехом, и хотя он эту статью одобрил и поддержал ее публикацию, тем не менее сказал: «Знаете, Андрей, Вы написали хорошую работу, но я Вам советую продвижение в этой области отложить до того времени, когда Вам будет где-то ближе к 40 годам. А сейчас Вам нужно сосредоточить свое внимание на новых проблемах, на тех вещах, где путь к успеху не так очевиден и не так близок».

Пройдет время, и один, а может, и не один из многих учеников академика Андрея Петровича Ершова такими же или похожими словами охарактеризуют вклад своего учителя в отечественную и мировую науку.

И. КОВАЛЬСКИЙ,
член Союза журналистов СССР,
редактор-консультант ЦТ



С. В. СЕРГЕЕВ

ЭЛЕКТРОННЫЙ ДОМ

Современного человека трудно удивить достижениями микроэлектронной техники. О них охотно пишут газеты, вещает радио, их показывает телевидение. Различного характера сведения о сложнейших робототехнических комплексах, хитроумных станках и мощных ЭВМ постоянно проходят через наше сознание, не задерживаясь в нем надолго. Да это и понятно, ведь для большинства людей электроника и вычислительная техника представляются пока еще чем-то абстрактным, далеким, не касающимся вроде бы их самих.

Однако благодаря стечению целого ряда обстоятельств уже сегодня созрели предпосылки для непосредственного общения каждого человека со сложнейшими продуктами нашей электронной эры. Это произошло отнюдь не в результате широкой компьютерной грамотности населения и не как следствие очередного людского увлечения, просто мы уже подошли к тому этапу в своем развитии, когда сама электроника все настойчивее стучится к нам, в наш дом.

Некоторые электронные приборы уже давно и прочно заняли свое место в нашей жизни. Это, например, телевизор, электронные часы, телефоны с памятью. Другие еще только начинают проникать в наш быт: например, СВЧ-печь, персональный компьютер, электронная почта. Что же ждет нас завтра?

Эти и другие непростые вопросы, возникающие все чаще по мере ошеломляющего развития электроники, по мере возрастающего влияния компьютеризации на жизнь общества и человечества в целом, создают множество новых проблем, задумываться о которых мы раньше просто не могли. При этом многие проблемы мы создаем себе сами, не сумев вовремя заметить важные тенденции, принять правильные решения и воплотить их в жизнь.

Целям практического решения этих вопросов служат различные программы, ведущиеся во многих развитых странах, которые можно объединить под общим названием «электронный дом». Само название проблемы подчеркивает, что в ближайшие годы «электронная начинка» станет определяющим фактором повышения комфорта, привлекательности и экономичности человеческого жилища.

Первые практические разработки, относящиеся к этому направлению, появились в 70-х годах нашего столетия, и с тех пор в различных масштабах они ведутся постоянно во многих странах. Среди причин, вызывающих растущий интерес к проблеме «электронного дома», основной, несомненно, является бурный прогресс в развитии микроэлектроники и аппаратуры, созданной на ее базе, а также новые возможности в процессах автоматизированного управления устройствами и механизмами. Появление микропроцессоров стало революционным в деле широчайшего распространения недорогих, доступных каждому электронных устройств. Наметившаяся в электронной технике тенденция постепенного слияния средств связи и обработки данных глубоко затрагивает многие стороны нашей повседневной жизни. Электроника в виде «разумных» специализированных устройств стала проникать всюду — в домашнюю развлекательную аппаратуру, в электробытовые товары, автомобили, учреждения, магазины, больницы, школы. Во всех этих направлениях уже имеются значительные достижения, но перспективы будущего многим и сейчас покажутся фантастикой [1].

Другой важной причиной, стимулировавшей работы в области «электронного дома», стал энергетический кризис, охвативший мировую экономику в 70-е годы. Поиски новых источников энергии, эффективных способов ее экономии в условиях высокой комфортабельности жилища легли в основу многих интересных проектов. Наиболее удачным можно считать жилой дом,

построенный фирмой Sharp Corp. на территории ее центральной исследовательской лаборатории в г. Тенри (Япония) в качестве одной из частей проекта, финансируемого министерством внешней торговли и промышленности [2].

Энергия для этого здания поступает как от элементов, преобразующих солнечный свет непосредственно в электричество, так и от коллектора солнечного тепла, который используется для нагрева воды. В доме установлен также воздушный кондиционер нового типа, приводимый в действие тепловой энергией. В нем применены теплообменники, утилизирующие как свободную, так и скрытую теплоту воздуха. Еще один теплообменник служит для передачи тепла от нагреваемой солнцем воды к воздуху. Воздушный кондиционер применяется для охлаждения, обогрева и регулирования влажности. Часть воды, нагреваемой солнцем, используется для ванны.

Фотоэлектрическая система, состоящая из нескольких панелей, разделена на две части. Первая, обеспечивающая около 90% общей энергопроизводительности, снабжает энергией вентиляторы и насосы и используется в качестве вспомогательного источника питания для осветительной сети. Вторая, дающая остальные 10% энергии, питает главный радиоприемник, телевизор и все осветительные приборы в комнате. Она также заряжает аккумулятор напряжением 12 В и емкостью 500 А·ч.

Электронный информационный центр в доме управляет энергосистемой, поставляя жильцу такие данные, как температура наружного воздуха, уровень солнечной радиации, вырабатываемая мощность и температура резервуара коллектора тепла. Он также показывает температуру в помещениях, уровень и температуру воды в ванне. Но помимо этих функций усложненного термостата, информационный центр организует отображение на экране почтовых сообщений, объявляет о прибытии гостей, предупреждает о вторжении нежелательных лиц и о таких чрезвычайных событиях, как пожар.

Несколько иной подход к проблеме реализован в проекте «Агватуки». Такое же название носит и специальное здание, построенное близ Финикса (США). Одна из особенностей «Агватуки», рекламируемого в качестве «жилища будущего», — вычислительная система, разработанная отделением фирмы Motorola на базе ее микропроцессора 6800.

Система состоит из пяти микрокомпьютеров, называемых узлами, которые расположены в различных частях дома. Они разделяют между собою общую информационную нагрузку, а в их информационном обеспечении и архитектуре упор сделан на избыточность, чтобы снизить нежелательные эффекты при выходе из строя части аппаратуры. В системе имеется даже связной процессор, обеспечивающий связи между узлами системы. Ввод и вывод информации осуществляются от клавиатуры и телевизионного приемника. Информация хранится в накопителях на гибких дисках.

У вычислительной системы «Агватуки» пять функций: информация (передача сообщений), переключение электрических нагрузок, контроль параметров окружающей среды, управление расходом энергии и обеспечение безопасности жилища. Информационная база подразделяется на два типа массивов: в одном файле хранится информация в виде записей, например банковских уведомлений, другой служит в качестве календаря, хранящего важные даты.

Система управления перераспределяет нагрузку между электропотребителями дома. Вопрос о том, какие условия должны определять включение и выключение тех или иных нагрузок, решает хозяин дома. Например, осветительные приборы включаются и выключаются настенными выключателями, но владелец может задать такой режим, когда компьютер будет включать освещение автоматически с наступлением сумерек или определенного времени дня или в том случае, если детекторы движущихся объектов обнаружат присутствие кого-либо в комнате. Таким же образом можно включать и выключать электрические и элект-

ронные аппараты, присоединенные к электросети через стенные розетки [4].

Контроль параметров окружающей среды несколько сложнее, чем в обычном доме, поскольку «Агватуки» разделен на три зоны — внутренний двор, закрытые помещения и гараж, в каждой из которых осуществляется отдельное регулирование температуры. Вычислительная система решает не только когда нагревать или охлаждать каждую зону, но и какими средствами. Она запрограммирована так, чтобы можно было выбрать самый дешевый из возможных способов. Так, если летним вечером во внутреннем дворике стало слишком тепло, то компьютер выбирает, что лучше: открыть окна (предварительно определив наружную температуру), включить систему испарительного охлаждения, предпочтительную в условиях пустыни, или включить воздушный кондиционер.

Управление расходом энергии в «Агватуки» — это главным образом информационный процесс. Система следит за общим потреблением энергии, контролируя расходы на отдельных нагрузках. Владелец дома может проверить уровень мощности на каждой нагрузке и даже получить сообщения по телефону, подключенному к принтеру за пределами дома.

Система безопасности реагирует на сигналы от датчиков дыма и движущихся объектов. Такие датчики расположены в каждой комнате. Система может также взаимодействовать с другой системой—управления освещением, включая осветительные приборы и выключая их, когда в доме никого нет. Кроме того, программа безопасности «Агватуки» управляет дверями дома, в которых отсутствуют традиционные замки и запоры и соответственно замочные скважины. Для того чтобы открыть дверь, необходимо набрать соответствующий код на небольшой клавиатуре у входа. Владелец дома может по желанию изменять этот код и даже назначать определенные коды на определенные дни недели.

Если система безопасности обнаруживает опасные условия, ее реакция будет зависеть от серьезности ситуации. Например, пожар приведет к включению звуковой сигнализации в доме, но компьютер может быть запрограммирован и на посылку заранее записанного сообщения в местную пожарную часть. Если же нарушение незначительное: например, кто-то набрал на двери неправильный код — система просто печатает сообщение об этом на принтере владельца.

Как отмечают специалисты, главное условие успеха любого проекта дома будущего заключается в легкости программирования используемых систем. Жильцы «электронных домов» не должны быть программистами, чтобы получать удовольствие от жизни в среде, насыщенной электронными устройствами. Наличие электронной системы не должно обременять их — она будет «прозрачной» для хозяев дома.

Кроме отдельных экспериментальных проектов человеческого жилища будущего, в целом ряде стран уже сейчас реализуются проекты жилых и административных зданий, которым присущи многие рассмотренные выше свойства.

Например, новое здание аэровокзала в Олбани-Каунти (штат Нью-Йорк, США) благодаря использованию оригинальных архитектурных и технических решений, соблюдению новых национальных стандартов на материалы, осветительное, вентиляционное и другие виды оборудования, широкому внедрению средств автоматизации, кроме многих своих достоинств, оказалось значительно более экономичным с точки зрения потребления энергии [5].

Световой фонарь в верхней части здания аэровокзала обеспечивает 40% всей потребности в освещении и 20% теплообеспечения за счет поступления солнечного излучения. Микрокомпьютер, запрограммированный до 2000 г., регулирует положение жалюзи в зависимости от высоты солнцестояния и осуществляет постоянный контроль параметров воздуха внутри помещения и снаружи. В ясный зимний день солнечная энергия нагревает заднюю темную стену, несущую световой фонарь и каменный пол аэровокзала, создающий дополнительную теплоаккумулирующую массу. Нагретый воздух прогоняется через пространство за стеной и

циркулирует по всему зданию. В ночное время жалюзи, заполненные пенистым теплоизолирующим материалом, закрываются, чтобы сохранять тепло. Летом жалюзи отражают прямые солнечные лучи, но пропускают рассеянный свет. Теплый воздух собирается под световым фонарем и выводится наружу вентиляционной системой. При ярком дневном свете фотоэлектрические регуляторы уменьшают искусственное освещение, создаваемое экономичными люминесцентными и ртутными лампами. С точки зрения эксплуатационных расходов подобные здания самые экономичные при обеспечении заданного уровня комфортности жилища. Электронные управляющие системы, средства кондиционирования, освещения и автоматики в зависимости от конкретных погодных условий, времени суток, размеров жилых помещений и других факторов автоматически обеспечивают оптимальное потребление тепла, электроэнергии, горячей и холодной воды. Следовательно, можно с уверенностью говорить об экономической целесообразности создания и массового строительства подобных жилых комплексов уже в ближайшие 10—15 лет, особенно учитывая наметившуюся тенденцию к постоянному возрастанию стоимости потребляемой электроэнергии, газа и воды.

Подтверждением этого может служить положительный опыт строительства современных зданий, накопленный в целом ряде стран. Например, в США с момента первых проявлений нефтяного кризиса в начале 60-х годов объем валового национального продукта вырос на 35% без увеличения потребления энергетических ресурсов. Основная причина этого в том, что энергия для сервиса, т. е. расходуемая на создание комфортных условий в жилых и служебных помещениях, передвижения на всех видах транспорта и даже на охлаждение напитков в жаркую погоду, в настоящее время потребляется в гораздо меньших размерах, чем в 1973 г.

Сокращение энергопотребления в основном достигается более экономичным расходованием энергии в жилых и административных зданиях. В результате их оснащения новыми техническими устройствами и улучшения эксплуатации систем освещения, отопления и вентиляции общая сумма счетов за расход различных видов энергии в целом по стране сократилась на 45 млрд. долл. в год. Несмотря на то что количество квартиросъемщиков увеличилось на 20 млн., а прирост жилой и служебной площади составил 1,7 млрд. м^2, ежедневное потребление горючих продуктов на обогрев в пересчете на нефть снизилось почти на 200 тыс. т.

Размеры такой экономии не должны удивлять. Следует учесть, что здания являются крупнейшим потребителем энергии. В США на их долю приходится 40% всей потребляемой в стране энергии. Большая часть вырабатывающейся в стране электроэнергии идет на обеспечение зданий: в общей сумме счетов за электроэнергию, которая равна 150 млрд. долл., 75% составляют счета на оплату электроэнергии, использованной в зданиях.

Понятно, что без широкого использования основополагающих концепций жилища будущего в предстоящие 50—100 лет, пока будут существовать здания, построенные в 80-е годы, нашей стране придется дорого расплачиваться за принимаемые сегодня решения в области строительства.

Тем не менее вопросы обеспечения комфортных условий в доме при минимальных затратах в настоящее время как бы отходят на второй план. Наибольшее внимание сейчас уделяется возможностям телекоммуникации, основанным на базе домашних электронных систем и получения информационных услуг. Один из увлекательнейших вопросов, обсуждаемых сегодня, касается перспективы использования персонального компьютера. Как показывает опыт создания экспериментального дома будущего, большинство применений личного компьютера будет незаметным для потребителя, которому не надо будет знать, что компьютер работает. Сами вычисления, по сути дела, будут наименее важной функцией этих устройств, занимая последнее место после хранения и поиска данных и операций связи. Возможно, поэтому компьютер в большей степени будет терминалом для связи с

миром внешней информации, нежели устройством обработки данных.

В заключение можно сделать следующие выводы. Работы в области создания «жилища будущего», ведущиеся сейчас во многих развитых странах мира, выявили ряд тенденций, позволяющих судить о том, каковы будут в общем черты нашего дома около 2000 г. Электроника обещает оказать огромное воздействие на средства контроля внутренней среды такого жилища и его связи с внешним миром. Проблемы «электронного дома» будущего не являются самоцелью или утопической мечтой. Это в экономическом и социальном плане выгодное решение одной из проблем повышения жизненного уровня народа полностью соответствует потребностям и возможностям ближайшего будущего.

Литература

1. Микропроцессоры. Кн. 1. Архитектура и проектирование микро-ЭВМ. Организация вычислительных процессов. Нестеров П. В. и др./ Под ред. Л. Н. Преснухина. — М.: Высшая школа, 1986.

2. Громов Г. Р. Национальные информационные ресурсы: проблемы промышленной эксплуатации. — М.: Наука, 1984.

3. Электроника: прошлое, настоящее, будущее. — М.: Мир, 1980.

4. Како Н., Эманэ Я. Датчики и микро-ЭВМ. — Л.: Энергоатомиздат, 1986.

5. Энергоэкономичные здания. Артур Р. Розенфельд, Дейвид Хафмейстер //В мире науки. — 1988. — № 6.

6. Француз у экрана «Минитела». Королев Ю. //Эхо планеты. — 1989. — № 5.

Город — компьютер

Специальный корреспондент газеты «Комсомольская правда» в Токио сообщает.

К 2000 г. близ Токио, в префектуре Тиба появится первый на планете полностью компьютеризированный город. Об этом объявил автор проекта — профессор Токийского университета Кэн Сакамура. Спонсоры — 130 корпораций, включая такие всемирно известные, как «Фудзицу», «Сони», Ай-би-эм, «Моторола», согласны на паях предоставить необходимые для осуществления замысла 100 миллиардов иен.

Что значит полностью компьютеризированный? Ну, к примеру, освещение в домах будет регулироваться автоматически, в зависимости от того, ясный день или пасмурный, полдень или сумерки. В единую и легкоуправляемую систему будут объединены ванная, телефон, музыкальная и видеоаппаратура, кухонные агрегаты. Жителю чудо-города не придется утруждать себя таким утомительным делом, как поворачивание водопроводного крана. Достаточно будет нежного прикосновения к стене. И даже, извините, воду в туалете спускать необязательно — за этим проследит компьютер.

Офисы, размещаемые в городе, будут полностью освобождены от бумажного делопроизводства. Не понадобятся ни курьеры, ни секретари. Нужный служащему файл нажатием кнопки будет за несколько секунд доставлен из подвального хранилища, либо необходимая справка появится на дисплее рабочего стола. Деревья будут расти не только снаружи, но и внутри зданий.

Город-компьютер по площади будет невелик — около одного квадратного километра. Население тоже, понятно, ограниченным — тысяча постоянных жителей и 6 тысяч тех, кто будет ежедневно приезжать сюда на работу. Кто они — этот вопрос авторы проекта пока обходят молчанием, хотя было бы небезынтересным узнать предполагаемую «номенклатуру».

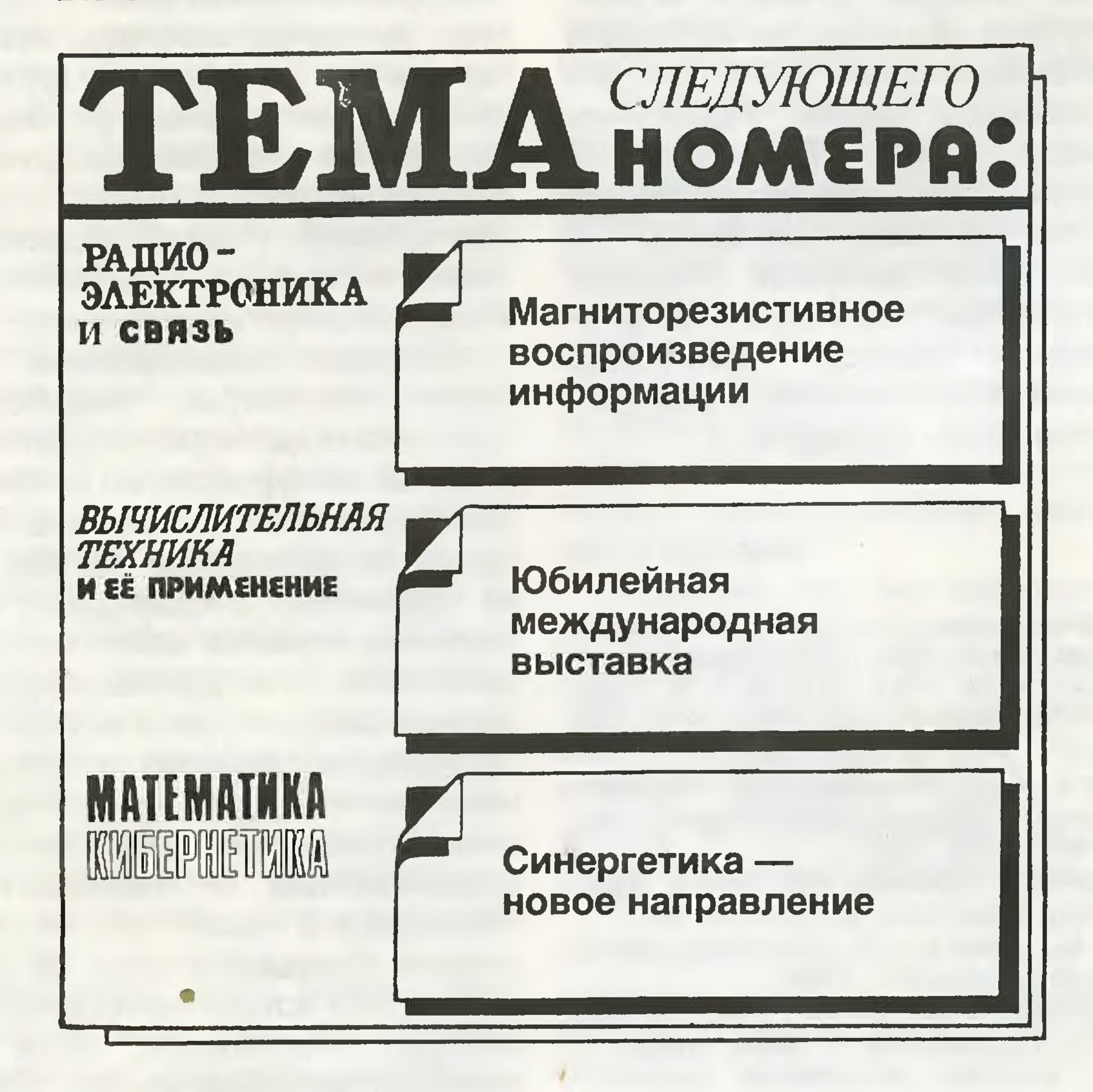
«Комсомольская правда», 1989, 14 мая

Уроки программирования. — М.: Знание, 1989.
У71 — 48 с. — (Новое в жизни, науке, технике. Сер. «Вычислительная техника и ее применение»; № 10).
ISBN 5-07-000923-0
20 к.

Материал этого выпуска составлен по урокам программирования, которыми пользуются учащиеся в заочной школе молодого программиста (Вильнюс). Цель выпуска — познакомить читателя с общими принципами программирования, научить читать и самим составлять программы. При этом научиться программировать могут и те, у кого нет возможностей работать с компьютером. Представленного в книжке материала достаточно для самостоятельного составления программ небольших задач. Программы составлены на языке высокого уровня ПАСКАЛЬ.

Брошюра рассчитана на широкий круг читателей.

2404010000 **ББК 32.973-01**



Научно-популярное издание

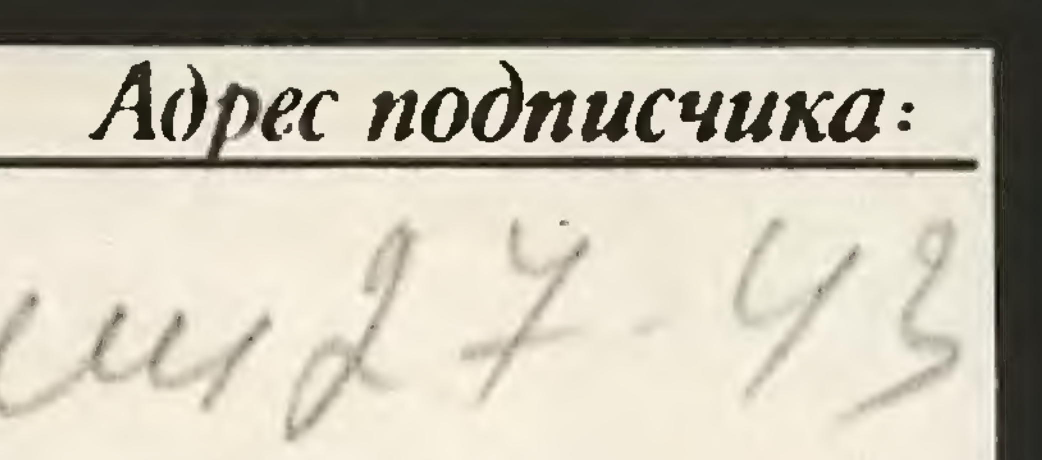
УРОКИ ПРОГРАММИРОВАНИЯ

Гл. отраслевой редактор *Л. А. Ерлыкин*
Зам. гл. отраслевого редактора *Г. Г. Карвовский*
Редактор *Б. М. Васильев*
Мл. редактор *Н. А. Васильева*
Художник *В. Н. Конюхов*
Худож. редактор *М. А. Гусева*
Техн. редактор *А. М. Красавина*
Корректоры *В. И. Гуляева, Л. И. Иванова, Е. А. Шарикова*

ИБ № 10277

Сдано в набор 29.06.89. Подписано к печати 30.08.89. Т-01490. Формат бумаги $70 \times 100^1/_{16}$. Бумага офсетная № 2. Гарнитура журнально-рубленая. Печать офсетная. Усл. печ. л. 3,90. Усл. кр.-отт. 8,45. Уч.-изд. л. 4,48. Тираж 64717. Заказ 418. Цена 20 коп. Издательство «Знание». 101835, ГСП, Москва, Центр, проезд Серова, д. 4. Индекс заказа 894710. Ордена Трудового Красного Знамени Калининский полиграфический комбинат Государственного комитета СССР по печати. 170024, г. Калинин, пр. Ленина, 5.

Цена 20 коп. Индекс 70195





Подписная научно-популярная серия

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Грамотность и программирование не только выстраиваются в параллель, соединяясь мостиками аналогий, но и дополняют друг друга, формируя новое представление о гармонии человеческого ума.

А.П. Ершов

Язык в совокупности с программным обеспечением, которое "понимает" его, может полностью преобразить ЭВМ, превратить ее в машину совершенно другого типа.

Лоуренс Г. Теслер

Наш адрес: СССР, Москва, Центр, проезд Серова, 4